Universidade de Aveiro
2014
Departamento de Eletrónica, Telecomunicações e Informática

Denis Ryazanov

Recolha e organização de informação de multimédia


Universidade de Aveiro
2014
Departamento de Eletrónica, Telecomunicações e Informática

Denis Ryazanov

Recolha e organização de informação de multimédia

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor Doutor Joaquim Arnaldo Carvalho Martins, Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho à minha família.

o júri

presidente

Professor Doutor Joaquim Manuel Henriques de Sousa Pinto
Professor Auxiliar da Universidade de Aveiro

Vogais

Professor Doutor Álvaro Pedro de Barros Borges Reis Figueira
Professor Auxiliar da Faculdade de Ciências da Universidade do Porto

Professor Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro

agradecimentos

Ao meu orientador, Professor Doutor Joaquim Arnaldo Carvalho Martins e ao Mestre Marco Pereira pela disponibilidade, motivação e todo o apoio que me deram ao longo deste trabalho.

À minha família pelo apoio e força que me deu ao longo do curso em momentos difíceis, sem qual não conseguia concluir esta etapa da vida.

À minha namorada pela paciência, compreensão e apoio que deu durante este ano.

Aos meus colegas pela ajuda, amizade e momentos de convívio que tornaram estes anos inesquecíveis.

palavras-chave

repositório de conteúdo multimédia, MVC, aplicação web, Grails, Groovy.

resumo

Nos últimos anos a produção de conteúdos multimédia com recurso a ferramentas digitais tem aumentado. Esse facto implicou a aparência de muitas aplicações e serviços que tem como a função o armazenamento desse conteúdo. No entanto, cada aplicação e serviço têm o seu objectivo específico, o que obriga o utilizador a usar várias soluções disponíveis conforme as suas necessidades, que por sua vez resultou na complexidade de gestão.

O principal objectivo desta dissertação é desenvolvimento de uma aplicação web com as várias funcionalidades, que permite gerir o conteúdo multimédia de vários tipos, local ou publicado nos outros serviços existentes, e partilhar esse conteúdo em forma de uma galeria pública. A aplicação tem integração com os outros serviços existentes, suporta as pesquisas por *tags*, título ou metadados dos ficheiros e permite personalizar a galeria pública com os vários *templates* e *layouts*.

Para o desenvolvimento da aplicação foi usado *framework Groovy/Grails*, orientado ao desenvolvimento ágil de aplicações web e compativel com *Java*, usando o repositório de conteúdo *Apache Jackrabbit*.

keywords multimedia content repository, MVC, web application, Grails, Groovy..

abstract

Multimedia content production with the use of digital tools has increased in recent years. This required the appearance of many applications and services that have the function of storing content. However, every application and service has its specific objective. Furthermore, it obliges users to use various available solutions according to their needs and, as a result, give rise to the complexity of management.

The main objective of this thesis is the development of a web application with several functionalities, that allows to manage media content of different types, local or which has been published in other existing services, and share this content in the form of a public gallery. The application has integration with other existing services, supports searches by tags, title or metadata and allows customize the public gallery with different templates and layouts.

For application development was used framework Groovy/Grails, it is oriented to the agile development of web applications and compatible with Java, using the Apache Jackrabbit content repository.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos:

AJAX - Asynchronous Javascript and XML

API - Application Programming Interface

CMS - Content management system

CSS – Cascading Style Sheets

GSP – Groovy Server Pages

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

HQL - The Hibernate Query Language

IoC - Inversion of Control

JCR – Content Repository API for Java

JSR – Java Specification Request

JSON – JavaScript Object Notation

MVC – Model View Controller

ORM – Object Relational Mapping

SGBD – Sistema de Gestão de Base de Dados

SSL – Secure Sockets Layer

URL – Uniform Resource Locator

URI – Uniform Resource Identifier

XML – eXtensible Markup Language

WYSIWYG - What You See Is What You Get

1. Introdução

1.1 Enquadramento

As tecnologias estão a evoluir com uma velocidade enorme. Hoje em dia os dispositivos de criação de conteúdo multimédia estão acessíveis para qualquer pessoa. Uma máquina fotográfica ou de filmar deixou de ser um luxo e passou a ser um dispositivo que acompanha a maioria das pessoas no dia-a-dia. Quase todos os telemóveis modernos, os dispositivos que acompanham as pessoas no seu cotidiano, permitem gravação de vídeos, áudio, fotos ou até criação de documentos.

Isto resultou num aumento do volume de informação que necessita de ser guardada e no aumento da vontade de partilhar a informação com as outras pessoas. Ao responder às necessidades dos utilizadores, aparecem então as novas aplicações e serviços como galerias *online*, serviços *cloud* e sites de partilha de vídeos.

Deste modo os conteúdos dos utilizadores começaram a ficar mais *online*. Cada aplicação e serviço têm o seu objectivo, o que obriga o utilizador a usar vários serviços, conforme as necessidades, seja para partilhar os vídeos, criar galerias ou fazer *backup* dos ficheiros. Consequentemente, com o crescimento de número de serviços existentes, o utilizador esquece onde tem o seu conteúdo e acaba por perdê-lo.

Hoje em dia já existem algumas soluções para repositórios digitais pessoais, alguns disponíveis para o público e outros apenas como os projectos de investigação, alguns para suportar os múltiplos tipos de ficheiros e outros para os tipos de ficheiros específicos.

Para mim a fotografia é uma paixão. Seria importante não só ter todas as minhas fotografias disponíveis num sítio de maneira organizada, mas também ter a possibilidade de partilhá-las com os outros utilizadores. Assim como eu necessito de uma ferramenta para fotografias, os outros utilizadores podem ter a necessidade de organizar os outros tipos de ficheiros. Portanto, aparece a necessidade de uma ferramenta genérica.

1.2 Objectivos

O objectivo desta dissertação é desenvolver um serviço que é capaz de gerir o conteúdo multimédia de vários tipos e mostrar o conteúdo seleccionado para as pessoas. A aplicação deve dar a possibilidade de importar e exportar o conteúdo dos vários serviços existentes.

Também deve permitir organizar o conteúdo de maneira mais evidente possível. A solução possível seria a organização em forma de uma árvore de nós semelhante aos sistemas de ficheiros dos sistemas operativos mas que seja independente do sistema operativo.

Uma das funcionalidades importantes da aplicação é a possibilidade de marcar o conteúdo como privado ou público e posteriormente gerar uma galeria de conteúdo público para outros utilizadores.

Outra característica da aplicação deve ser a possibilidade de construir a galeria pública personalizada aplicando vários *templates* e *layouts* como num *Content Management System* (CMS), de maneira a permitir a cada utilizador da aplicação construir a própria galeria do seu conteúdo, juntando os módulos existentes ou implementando os próprios módulos em linguagem *HyperText Markup Language* (HTML).

Mais um ponto forte deve consistir em dar a possibilidade para o utilizador a utilizar a aplicação de maneiras diferentes consoante as suas necessidades e preferências. Ou seja o repositório pode ser uma galeria de fotografias de autor, uma biblioteca digital, ou até uma colecção de conteúdos de aprendizagem que contém os livros, os trabalhos dos autores que inspiram, os vídeo-*tutorials* do Youtube, etc.

Para o desenvolvimento da aplicação será utilizado o *framework* moderno para o desenvolvimento de aplicações Web Grails, que combina as melhores tecnologias relacionadas com Java. Para a reutilização futura da aplicação desenvolvida é possível guardar o projecto como *plugin* para Grails, que pode ser integrado nos outros projectos sobre repositórios digitais pessoais, que estão em desenvolvimento no grupo de trabalho e que usam esta tecnologia.

1.3 Estrutura da dissertação

A dissertação está dividida em 6 capítulos: Introdução, Estado da Arte, Análise de requisitos, Arquitectura, Resultados e Conclusões.

Introdução – Neste capítulo são descritos o enquadramento e os objectivos da dissertação.

Estado da Arte – Neste capítulo é realizada uma análise de aplicações e serviços existentes e feita a sua comparação. São apresentadas também as tecnologias utilizadas.

Análise de requisitos – Este capítulo começa pela descrição da visão geral do sistema, a seguir são descritos os casos de utilização e por fim são demonstrados e descritos os *mockups* da aplicação.

Arquitectura – Neste capítulo são descritos todos os detalhes da arquitectura da aplicação.

Resultados – Neste capítulo é demonstrado o resultado final e descrito o funcionamento da aplicação. São apresentadas algumas imagens com o aspecto final da aplicação e simuladas algumas das funcionalidades principais da aplicação.

Conclusões – Neste capítulo são descritas as conclusões e os trabalhos futuros do projecto.

2. Estado da Arte

2.1 *Content Management Systems*

Content Management System é um sistema de gestão de conteúdos. Os CMS são usados para a recolha e combinação de diferentes fontes de informação em uma única unidade. Permitem a criação, armazenamento e administração de conteúdos de forma dinâmica, através de uma interface de acesso via *browser*. A principal função desses sistemas é mostrar as páginas dos *sites* aos utilizadores, criando as páginas em tempo real a partir de conteúdo e *templates* predefinidos.

Deste modo pode-se dizer que não existe um *site* como um conjunto de páginas. Existe o *design* (*templates*) e separadamente o conjunto de conteúdos, como texto, imagens, documentos, etc. O CMS cria a página no momento de pedido do utilizador.

Hoje em dia no mercado existem muitos CMS. Alguns são orientados para tarefas específicas (blogs, lojas *online*, fóruns, etc) e outros são mais universais que permitem aos desenvolvedores criar praticamente qualquer solução. Alguns exemplos dos CMS mais populares são Joomla [42], Drupal [43], WordPress [44], Blogspot [45], Magnolia [46], Hippo CMS [47], etc.

2.2. Repositórios pessoais

Hoje em dia existem vários repositórios para armazenar o conteúdo dos utilizadores. Alguns deles são apresentados nos capítulos seguintes.

2.2.1 MyLifeBits

MyLifeBits [36] – é uma plataforma baseada em SQL que possibilita aos seus utilizadores gravar, armazenar e aceder ao arquivo pessoal. Através desta plataforma é possível armazenar o conteúdo e metadados de vários tipos, incluindo contactos, documentos, *emails*, eventos, fotos, músicas e vídeos. Além disso, o sistema monitoriza cada página web visitada, todas as sessões de *chat* de aplicações de *Instant Message* e todas as conversões telefónicas. É de notar que a MyLifeBits é uma experiência de investigação e não está disponível para o público.

O conteúdo pode ser interligado entre si implicitamente usando o tempo de criação, os dados de GPS incorporado, etc, ou explicitamente, usando as ligações como “pessoa em foto”, ligações entre o contacto e a foto, etc. Usando as ligações, a árvore de directórios tradicional pode ser substituída por colecções baseadas em grafos acíclicos dirigidos. Desta maneira qualquer objecto, incluindo a colecção, pode ser incluído em qualquer número de conjuntos de colecções-pais.

Uma vez que o conteúdo está na base de dados, é possível através de várias ferramentas incorporadas organizar, associar os metadados e aceder a informação.

A principal aplicação de MyLifeBits permite visualizar o conteúdo como a lista, grelha de *thumbnails* ou a *timeline*. Existe a possibilidade de comentar o conteúdo com o texto, voz ou outros ficheiros. A aplicação também tem a funcionalidade de *screensaver*, que visualiza aleatoriamente as fotografias e vídeos e dá oportunidade ao utilizador para comentar e avaliar os conteúdos.

2.2.2 PhotoGeo

PhotoGeo [37] é uma infra-estrutura que possibilita aos utilizadores armazenar, aceder e anotar as fotografias recorrendo aos novos algoritmos para anotação usando os metadados-chave como as pessoas que estão na foto (*who*), a localização onde foi tirada a foto (*where*), e a data e a hora quando foi tirada a foto (*when*).

PhotoGeo consegue aceder aos calendários como Google Calendar [44] para extrair a informação dos horários do utilizador na hora em que a fotografia foi tirada e a seguir obtém a informação geográfica com a ferramenta de mapeamento Wikimapia [45], dicionário geográfico GeoNames [46] e com o sistema fornecedor de localizações geográficas dos amigos do utilizador VadeMecum.

Wikimapia é uma ferramenta de mapeamento onde os utilizadores partilham os pontos de interesse em qualquer parte do mundo. PhotoGeo extrai a informação dos sítios onde as fotografias foram feitas no Wikimapia e obtém os metadados dos objectos que podem ser encontrados nesse ponto de interesse. Para identificar os nomes dos locais com mais precisão PhotoGeo usa dicionário geográfico Geonames. Com ajuda desse dicionário, PhotoGeo consegue anotar o nome de um edifício, uma montanha, uma estrada, uma rua ou um país. Deste modo os utilizadores não devem estar preocupados com as coordenadas. PhotoGeo extrai a informação de GeoNames a partir de *Web service* que recebe a localização da fotografia e retorna os nomes dos sítios associados à essa localização.

Para obter as localizações geográficas dos amigos do utilizador é usado o sistema VadeMecum, que é um sistema para dispositivos móveis que fornece as informações contextuais sobre uma determinada pessoa. A PhotoGeo pergunta sobre as localizações geográficas dos amigos do utilizador no tempo em que foi tirada a fotografia. A informação obtida é utilizada no algoritmo que lista as pessoas que têm mais probabilidades de estar na fotografia.

Existem dois tipos de clientes: *mobile* e *web*. Os clientes *mobile* são responsáveis por capturar as fotografias nos dispositivos móveis e anotá-las incluindo as pessoas presentes nas fotografias a localização geográfica onde a fotografia foi tirada. Depois disso as fotografias são enviadas para o servidor. O cliente web é responsável pela gestão e visualização das fotografias.

2.2.3 PictureLife

PictureLife [18] - é um serviço *online* de armazenamento de fotos e vídeos com a possibilidade de sincronização e envio para redes sociais. O principal destino deste serviço é criação de álbuns de fotos e vídeos. PictureLife consiste em três serviços: os clientes de sincronização para Windows e Mac, a aplicação web para organização e partilha e a aplicação para iOS com as funcionalidades de cliente de sincronização e a aplicação web.

A aplicação web permite visualizar o conteúdo em forma de grelha de *thumbnails*, onde os utilizadores podem saltar para data especifica apenas com um *click*.

A PictureLife inclui as galerias separadas para amigos e para a família. As imagens que estão marcadas como "Partilhadas com a família" são visualizadas instantemente no *stream* da família por utilizadores que foram designadas para tal.

2.2.4 Trovebox

Trovebox [21] – é um serviço que permite a integração de fotos de vários serviços e redes sociais num sítio. Permite importar as fotos do Facebook, Instagram e Flickr em poucos passos simples. Permite também fazer *upload* directamente do computador com a funcionalidade de *drag and drop*. Dá a possibilidade de criar álbuns e partilha-los nas redes sociais como Facebook e Twitter.

2.3 Serviços de armazenamento existentes

Hoje em dia no mercado existem muitos serviços que oferecem a possibilidade de guardar e partilhar os conteúdos *online*, como serviços de armazenamento do tipo *cloud* ou serviços de armazenamento e partilha de fotografias e vídeos. Em baixo segue uma descrição de alguns serviços em geral e alguns que são mais únicos em particular. Por fim segue uma comparação entre esses serviços e a aplicação que será desenvolvida durante esse trabalho.

2.3.1 Serviços de armazenamento Cloud

Os serviços de armazenamento *cloud* são baseados na sincronização de dados. É um modelo de armazenamento *online* onde os dados são guardados nos servidores remotos. A maioria dos serviços *cloud* tem *software* cliente *desktop*, ao instalar qual é criada uma pasta sincronizada. Os utilizadores podem criar as pastas partilhadas com os outros utilizadores. Os exemplos de serviços *cloud* mais populares são Dropbox [17], Google Drive [20] e Mega [30].

2.3.2 Serviços de armazenamento e partilha de imagens

Os serviços de armazenamento e partilha de imagens, principalmente fotografias e desenhos, foram criados para atender as necessidades de artistas de artes visuais para partilhar os seus trabalhos com os outros utilizadores. Os utilizadores registados podem submeter o seu conteúdo nos servidores remotos junto com a informação do ficheiro, como título, descrição e *tags*. Os serviços mais populares são Flickr [19] e 500px [39].

2.3.3 Serviços de armazenamento e partilha de ficheiros de vídeos

São serviços que consistem em armazenamento de vídeos. Permitem ao utilizador carregar o vídeo para o servidor remoto. Os outros utilizadores podem visualizar, mas não têm possibilidade de guardar o vídeo no seu computador. Os mais conhecidos serviços são Youtube [40] e Vimeo [41].

2.4 Análise comparativa

Neste parágrafo faz-se uma análise comparativa entre os repositórios pessoais e outros serviços semelhantes existentes.

Armazenamentos de Imagens:

Todos os serviços apresentados permitem armazenar as imagens de formatos mais comuns como jpeg, png, gif e bmp.

Armazenamento de Vídeos:

Todos os serviços apresentados, excepto PhotoGeo, permitem armazenar os vídeos, entretanto o Trovebox apenas tem suporte de vídeos nos planos mais caros.

Armazenamento de Documentos:

Dos serviços comparados, apenas os serviços de armazenamento de ficheiros *cloud* e o MyLifeBits permitem armazenamento de documentos. Os outros serviços apenas permitem armazenar os ficheiros de formatos específicos.

Importação de outros serviços:

É uma funcionalidade que permite importar directamente os conteúdos de outros serviços para a própria aplicação. Das aplicações analisadas apenas PictureLife e Trovebox e alguns dos serviços de armazenamento e partilha de imagens permitem importar o conteúdo dos outros serviços.

Exportação para outros serviços:

É uma funcionalidade que permite exportar o conteúdo da aplicação para outros serviços existentes. Apenas o PictureLife tem essa funcionalidade.

Galeria pública:

É uma funcionalidade que permite expor o conteúdo na forma de galeria pública, onde todos os utilizadores não registados na aplicação pudessem visualizá-la. Quase todos os serviços apresentados têm essa funcionalidade, excepto PhotoGeo e MyLifeBits.

Suporte de *Tags*:

É uma funcionalidade que permite adicionar as palavras-chave ao conteúdo para facilitar a pesquisa. Dos conteúdos analisados apenas os serviços de armazenamento de ficheiros *Cloud* não suportam os *tags.*

Pesquisa por Metadados:

Os metadados, como já foi dito, são as informações que estão armazenadas em ficheiros e servem para armazenar a informação sobre o ficheiro em questão. Pesquisa por metadados pode ser uma ferramenta muito forte e útil. No entanto dos serviços apresentados apenas PhotoGeo e MyLifeBits têm essa característica.

Suporte de conteúdos externos:

É uma funcionalidade que permite gerar o conteúdo que não está directamente armazenado na aplicação. Isto quer dizer que a aplicação apenas tem uma ligação virtual para o ficheiro que está armazenado no outro serviço *online*. Esta funcionalidade é muito útil, pois o utilizador consegue gerir o conteúdo como se ele fosse armazenado na própria aplicação, mas sem gastar o espaço adicional do repositório. Nenhum dos serviços analisados tem essa característica.

Organização de conteúdo:

É a possibilidade de organizar o próprio conteúdo de maneira de utilizador. Nos serviços de armazenamento de ficheiros tipo *cloud* normalmente o conteúdo é organizado recorrendo as pastas. Nos serviços de armazenamento e partilha de imagens, no PictureLife e no Trovebox são criados os álbuns. O PhotoGeo usa as colecções, o MyLifeBits usa as colecções baseadas em grafos acíclicos dirigidos. E nos serviços de armazenamento e partilha de vídeos o conteúdo é organizado por *playlists* ou não tem qualquer organização.

Personalização da galeria:

É a possibilidade de personalizar a própria galeria. No Trovebox é possível mudar as cores da galeria nos pacotes mais caros. Em alguns serviços de armazenamento e partilha de imagens ou vídeos, como Flickr ou Youtube, é possível personalizar apenas a capa da galeria. Já os outros serviços não oferecem qualquer possibilidade de personalizar a galeria com o gosto do utilizador.

Adição de funcionalidades próprias:

É possibilidade de adicionar os próprios módulos à galeria pública. Nenhum serviço analisado possui essa funcionalidade.

	Armazenamento de ficheiros *Cloud*	Armazenamento e partilha de imagens	Armazenamento e partilha de vídeos	Picture Life	Trove box	MyLife Bits	Photo Geo	Repositório de Conteúdo Desenvolvido
Imagens	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Vídeos	Sim	Alguns	Sim	Sim	Sim	Sim	Não	Sim
Documentos	Sim	Não	Não	Não	Não	Sim	Não	Sim
Import	Não	Alguns	Não	Sim	Sim	Não	Não	Sim
Export	Não	Não	Não	Sim	Não	Não	Não	Sim
Galeria pública	Alguns	Sim	Sim	Sim	Sim	Não	Não	Sim
Suporte de Tags	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Pesquisa por Metadados	Não	Não	Não	Não	Não	Sim	Sim	Sim
Suporte de conteúdos externos	Não	Não	Não	Não	Não	Não	Não	Sim
Organização	Pastas	Álbuns	Playlist	Álbuns	Álbuns	Colecções	Colecções	Árvore Hierárquica
Personalização da galeria	Não	Alguns	Não	Não	Personalização das cores	Não	Não	Vários *templates* e *layouts*
Funcionalidades próprias	Não	Não	Não	Não	Não	Não	Não	Sim

Tabela 1 - Comparação de serviços de armazenamento existentes

2.5 Tecnologia usada

2.5.1 Padrão *Model-View-Controller* (MVC)

O padrão *Model-View-Controller* padrão separa a aplicação web em três partes funcionais distintas: modelo de dados (*Model*), interface do utilizador (*View*) e uma lógica de controlo (*Controller*). Deste modo, as alterações feitas num dos componentes têm menor impacto possível em outros componentes.

Neste padrão, o modelo de dados não depende do controlador ou interface, o que permite a criação do modelo de dados como um componente independente, por exemplo criar vários interfaces para o mesmo modelo.

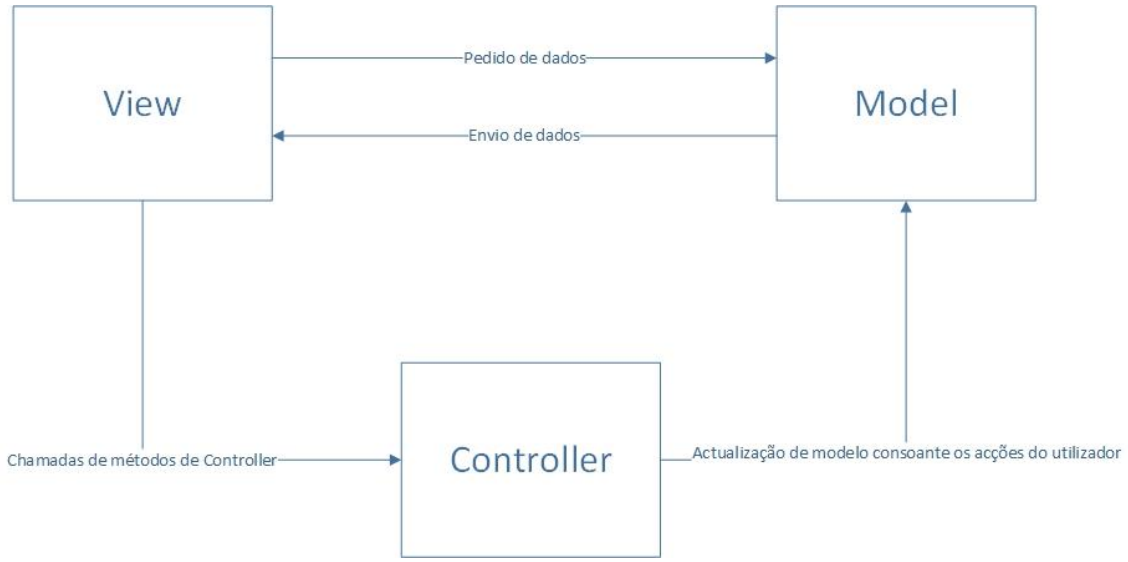

Figura 1 - Esquema de padrão Model-View-Controller

Na figura 1 podemos ver o esquema de padrão MVC, onde o *Model* representa os dados, *View* representa a interface do utilizador e o *Controller* proporciona a interacção entre o *Model* e o *View*.

Pode-se representar a sequência típica de funcionamento das aplicações que estão construídas com o padrão MVC de seguinte maneira:

1. Quando o utilizador entra na aplicação *Web*, o *script* de inicialização cria uma instância de aplicação e executa-o. Isso apresenta uma página, por exemplo a página inicial da aplicação.

2. A aplicação recebe um pedido de utilizador e identifica o *Controller* e o método solicitados.

3. A aplicação cria uma instância de *Controller* e corre o método, que pode chamar as funções de *Model* que retraem a informação da base de dados.

4. Depois disso, o método forma uma representação de dados obtidos do *Model*. A seguir gera e representa o resultado para utilizador.

2.5.2 Groovy

Groovy [1] - é uma linguagem de programação orientada aos objectos, projectada como um complemento para a linguagem Java com as capacidades de Python, Ruby e Smalltalk.

Groovy utiliza uma sintaxe semelhante à Java com a compilação dinâmica para *bytecode* Java e funciona directamente com outros códigos e bibliotecas Java. Entretanto a sintaxe do Groovy é mais flexível e poderoso.

A maior vantagem de Groovy consiste no facto que qualquer programa escrito em Java é um programa escrito em Groovy, ou seja não é necessário reescrever todo o código Java para Groovy, pois todas as bibliotecas de Java escritas nos últimos 15 anos podem ser usadas em Groovy.

Nas figuras 2 e 3 podemos ver dois códigos semelhantes, um é escrito em Java e outro em Groovy.

```
public class OlaMundo{
  public static void main(String[] args){
    System.out.println("Ola Mundo!");
  }
}
```

Figura 2 - Programa "Ola Mundo" em Java

```
println "Ola Mundo!"
```

Figura 3 - Programa "Ola Mundo" em Groovy

Eles representam a aplicação “Ola Mundo”, o exemplo mais básico de programação, que imprime na consola a mensagem “Ola Mundo!”. O primeiro exemplo (Figura 2) mostra quantas funcionalidades é necessário saber para perceber como funciona esse código. Primeiro tem que ser criado o ficheiro com o nome “OlaMundo.java”, que precisa de coincidir com o nome da classe para poder compilar. Além disso, para os novos utilizadores logo surgem muitas dúvidas, “porque é *public*?”, “o que é *static*?”, “o que é *void*?”, “porque a função tem nome *main*?”, etc.

O segundo exemplo (Figura 3) é um código 100% funcional e equivalente ao primeiro exemplo. Neste caso todos os detalhes, que não estão directamente relacionados com a tarefa em questão, estão escondidos atrás de uma única linha de código.

2.5.3 Grails

GRAILS [2] - é um *framework* para construção de aplicações Web escrito em linguagem Groovy.

É baseado no padrão MVC. O Grails foi concebido para atrair o interesse dos utilizadores na plataforma Java e fornecer aos desenvolvedores em Java a possibilidade de construir rapidamente aplicações Web com facilidade e flexibilidade que não estavam disponíveis antes.

O factor pelo qual o Grails é melhor dos seus concorrentes é escolha de projectos usados no seu desenvolvimento, aproveitando os *frameworks* testados e confiáveis.

O Grails assenta nalgumas das mais populares e melhores tecnologias de código aberto:

-Hibernate [3] – *framework* para mapeamento objecto-relacional (ORM) no mundo de Java.

-Spring [4] – *container* popular de inversão de controlo para Java de código aberto.

-SiteMesh [5] – *framework* robusto de renderização de *layouts*.

-Tomcat [6] – *container* de *servlets* para Java.

-H2 [7] – é um sistema de gestão de base de dados escritos em Java.

A grande vantagem de Grails é a integração e envolvimento desses *frameworks* através de introdução de outra camada de abstracção por meio da linguagem Groovy. O desenvolvedor a fazer uma aplicação em Grails não tem noção que está a construir uma aplicação de Spring ou Hibernate. Não é necessário tocar uma única linha de Hibernate ou ficheiro de eXtensible Markup Language (XML) de Spring, mas eles estão ali no caso se precisar deles. A figura 4 ilustra como Grails se refere a esses *Frameworks*.

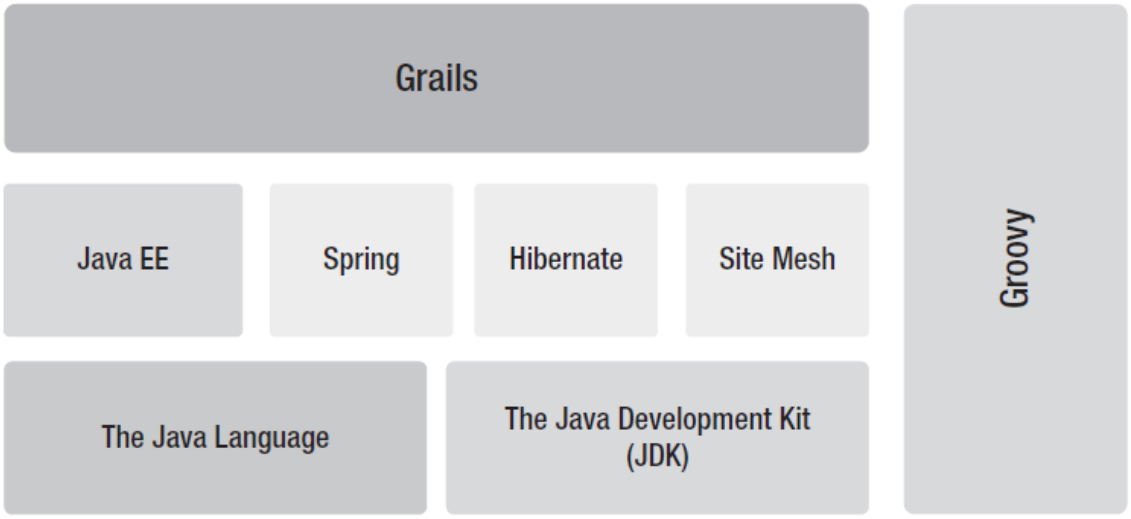

Figura 4 - Integração de vários *frameworks* com Grails e Groovy

2.5.3.1 Estrutura do projecto Grails

Após a criação do novo projecto, o Grails constrói uma estrutura de directórios necessários para a aplicação. A figura 5 mostra a estrutura de directórios do projecto acabado de ser criado.

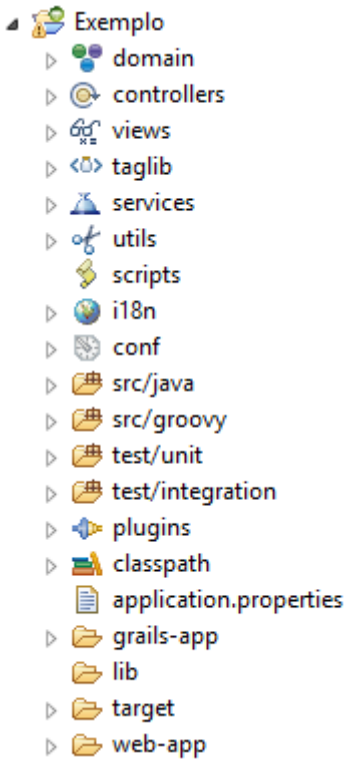

Figura 5 - Estrutura de directórios de projecto em Grails

Em baixo segue uma pequena descrição dos directórios principais:

domain – contém os modelos de dados

controllers – contém os controladores

views – contém as *views* e *templates*

taglib – contém as bibliotecas de *tags* criados pelo utilizador

services – contém os serviços

i18n – contém as mensagens suportadas pelas várias idiomas

conf – contém as configurações do projecto, como configurações de compilação e execução do projecto, definição de *plugins* e dependências, configuração de recursos, configuração de base de dados, configuração de mapeamentos etc.

web-app – contém os recursos da aplicação web, como imagens, ficheiros de *Cascading Style Sheets* (CSS) e *javascript*.

2.5.4 JCR (Java Content Repository)

Repositórios permitem dar acesso para leitura, escrita e pesquisa de dados independentemente das aplicações que necessitam desses dados.

De acordo com as especificações JSR-283 [8] e JSR-333 [9], o repositório consiste numa base de dados orientada a objectos que fornece armazenamento, pesquisa e recepção de dados hierárquicos. Além disso, a Interface de Programação de Aplicações (API) fornecida também permite controlo de versões de dados, transacções, controle das alterações, importação e exportação de XML, e até armazenamento de dados binários e metadados.

O repositório está organizado em forma de uma árvore de nós que têm propriedades. Eles estão endereçados pelo caminho semelhante aos sistemas operativos. Toda a informação é armazenada nas propriedades de nós e pode ser de vários tipos de dados, como String, boolean ou até dados binários para armazenamento de ficheiros.

Com a chegada de versão 2, o repositório funciona com a linguagem SQL2.

2.5.4.1 Apache Jackrabbit

Apache Jackrabbit [10] é um repositório de conteúdos de código aberto para a plataforma Java, que segue as normas JSR-170 [38] e JSR-283.

O repositório armazena o conteúdo hierarquicamente com suporte de conteúdo estruturado e não estruturado, pesquisa de texto, controlo de versões, transacções, observações e etc.

Usa o motor de pesquisa Apache Lucene [11] para as pesquisas de texto completas no repositório.

2.5.5 Metadados

Metadados consistem em informações que estão armazenadas em praticamente qualquer tipo de ficheiro. Servem para armazenar a informação sobre o ficheiro que a contém.

Podem incluir nome do autor, nome da empresa, nome do computador, data de criação e hora da última modificação, o número da versão, etc.

2.5.5.1 Formatos de Metadados

Os metadados têm vários formatos associados a vários tipos de ficheiros. O formato de metadados consiste num padrão para descrição formal de uma categoria de recursos. Esse

padrão normalmente inclui um conjunto de campos (atributos, propriedade, etc) que permite caracterizar o objecto em questão. Em baixo segue a descrição de alguns dos mais conhecidos formatos de metadados.

EXIF

Exchangeable Image File Format (EXIF) [10] - é uma especificação que permite adicionar às fotos digitais a informação adicional com a descrição da fotografia.

A informação guardada no EXIF pode ser usada pelo homem e por uma variedade de dispositivos (por exemplo por uma impressora, para impressão directamente da câmara digital).

Qualquer câmara digital grava no EXIF as informações detalhadas sobre a imagem, como o nome da câmara, as configurações da câmara com que são feita a foto, data de criação, etc. A figura 6 ilustra o exemplo de EXIF guardado pela câmara fotográfica digital Canon 550D.

Property	Value
Date taken	24/07/2014 17:39
Dimensions	3456 x 5184
Size	21,2 MB
Authors	Denis Ryazanov
Camera maker	Canon
Camera model	Canon EOS Kiss X4
Camera serial number	2213307756
ISO speed	ISO-100
F-stop	f/2.8
Exposure time	1/2000 sec.
Exposure bias	0 step
Exposure program	Manual
Metering mode	Pattern
Flash mode	No flash, compulsory
Focal length	50 mm
Lens maker	
Lens model	

Figura 6 - EXIF de uma fotografia tirada com câmara digital

MARC

Machine Readable Cataloging (MARC) [13] – é um conjunto de formatos para descrever os recursos bibliográficos. Existem diferentes tipos desse formato. Assim nos EUA e no Reino Unido é utilizado formato MARC 21, e na Europa e na Ásia é utilizado UNIMARC. A UNIMARC por sua vez é subdividida em expansões nacionais. Por exemplo JapanMARC no Japão ou RusMARC na Rússia.

DCMI

The Dublin Core Metadata Initiative (DCMI) [14] – é um dos formatos de metadados mais usados na internet para descrever os recursos de qualquer tipo (os ficheiros ou os objectos físicos reais).

ID3

ID3 [15] – é o formato de metadados mais usado em ficheiros de áudio em formato MP3. ID3 contém os campos com o título da música, nome do álbum, nome do artista e outras informações sobre o ficheiro que são usados por leitores de áudio para visualizar a informação do ficheiro.

2.5.5.2 Apache TIKA

Apache TIKA [16] é um conjunto de ferramentas multiplataforma escritas em Java que consegue detectar e extrair os metadados dos vários ficheiros usando as bibliotecas de análise sintáctica existentes. Tika junta essas bibliotecas para um único interface que permite fazer análise sintáctica de mais que mil formatos de ficheiros diferentes.

3 Análise de requisitos

3.1 Visão geral do sistema

Após uma pequena análise de requisitos, foi concluído que a aplicação além de juntar todas as características e funcionalidades das aplicações estudadas deve possuir mais algumas funcionalidades importantes.

Muitas das aplicações e repositórios pessoais estudados apenas permitem armazenar um tipo de ficheiro, portanto em primeiro lugar a aplicação deve ser mais genérica, ou seja deve permitir o armazenamento dos principais formatos de ficheiros multimédia.

A aplicação deve permitir fazer *import* e *export* do conteúdo de outros serviços, como o Flickr, Instagram, Youtube, etc e, ao contrário dos outros serviços e repositórios, deve permitir adicionar e gerir o conteúdo virtual, cujo ficheiro binário se encontra *online*.

Além disso a aplicação deve ter boa usabilidade. Ela precisa de ser simples e intuitiva. Qualquer pessoa, independente dos conhecimentos informáticos, deve facilmente e rapidamente navegar na aplicação.

Ao contrário dos repositórios estudados, cada utilizador deve ter a possibilidade de usar a aplicação de maneira mais adequada às suas necessidades. Pode criar o repositório pessoal privado, pode ter uma galeria pública de conteúdos para expor o seu conteúdo para os outros utilizadores, pode fazer uma biblioteca *online*, etc.

Outra característica que deve diferenciar a aplicação dos repositórios estudados é possibilidade de cada utilizador a personalizar a galeria pública consoante o seu gosto. Isto é não só a possibilidade de mudar o *layout*, mas também é possibilidade de criar os próprios módulos baseados no código HyperText Markup Language (HTML) e JavaScript e adicioná-los aos *templates* existentes para construir o *template* final da galeria pública.

A organização de conteúdos na aplicação deve ser flexível e evidente, cada utilizador pode criar a própria hierarquia de conteúdos consoante as suas preferências e necessidades.

A aplicação deve permitir a pesquisa de conteúdos por *tags*, título ou outros metadados quaisquer que podem estar incluídos no ficheiro.

Para guardar todo o conteúdo vai ser usado o repositório de conteúdo Apache Jackrabbit, que permite a independência dos sistemas operativos e bases de dados.

3.2 Casos de uso

A aplicação tem dois tipos de utilizadores. Por um lado são os utilizadores registados que gerem o seu repositório de conteúdos. Eles podem adicionar os conteúdos multimédia,

organizá-los hierarquicamente, importar o conteúdo de outros serviços, personalizar o *template* da sua galeria, etc. Por outro lado utilizadores que visualizam as galerias dos outros.

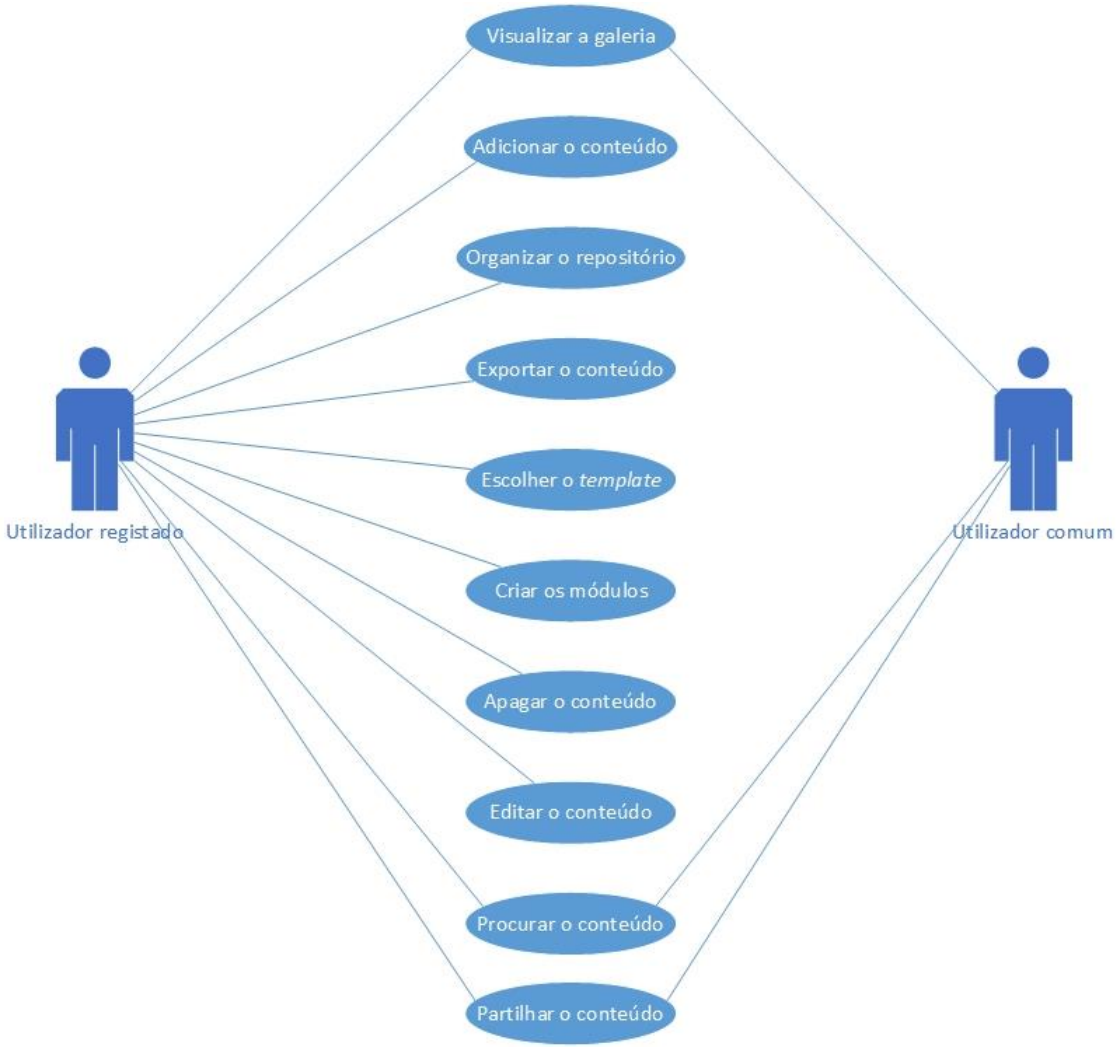

Figura 7 - Diagrama de casos de uso

-**Visualizar a galeria** – visualizar o próprio conteúdo que se encontra no repositório no caso de utilizador registado, ou visualizar o conteúdo que está marcado como público no caso de utilizador comum.

-**Organização do repositório** – o utilizador pode criar uma árvore hierárquica de nós e organizar o conteúdo conforme essa árvore.

-**Adicionar o conteúdo** – o utilizador pode fazer *upload* do PC do conteúdo multimédia para repositório ou importar dos outros serviços.

-**Apagar o conteúdo** – o utilizador pode apagar o conteúdo seleccionado.

-**Editar o conteúdo** – o utilizador pode editar o conteúdo, mudando o nome ou descrição, adicionando os *tags* e categorias.

-**Exportar o conteúdo** – o utilizador pode exportar o conteúdo para outros serviços.

-**Escolher o *template*** – o utilizador pode escolher um dos vários *templates* disponíveis.

-**Criar módulos** – o utilizador pode criar módulos próprios para inserir no *template* escolhido.

-**Procurar o conteúdo** – o utilizador pode procurar o conteúdo por *tags*, nome ou até metadados específicos.

-**Partilhar o conteúdo nas redes sociais** – o utilizador pode partilhar o conteúdo nas redes sociais.

3.3 Mockups e funcionalidades

Mockup é uma apresentação simples e pouco detalhada da futura aplicação que inclui todos os principais blocos de conteúdo. Permitem dar ideia das principais funcionalidades e organização visual da aplicação.

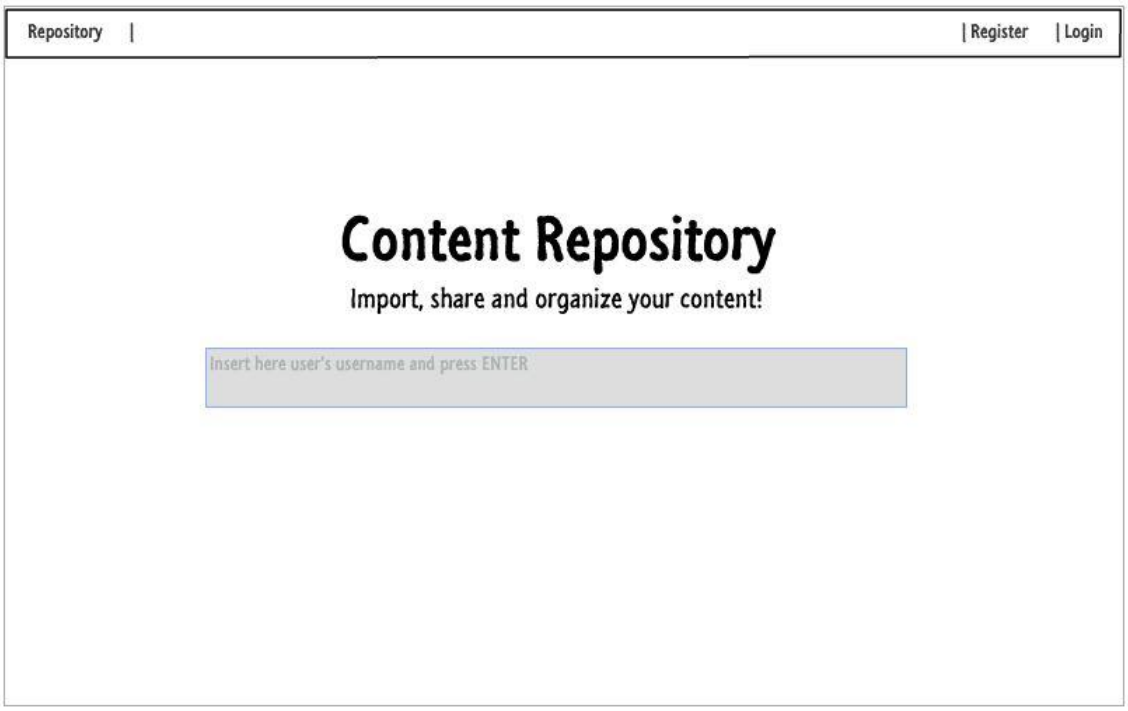

Figura 8 - *Mockup* do ecrã inicial

O primeiro *mockup* apresentado (Figura 8) corresponde à janela inicial que o utilizador vê quando entra no site. Tem *interface* simples e contém apenas o nome do repositório, botão para fazer *Login* e uma caixa de *input* para introduzir o nome do utilizador para entrar na galeria pública dele.

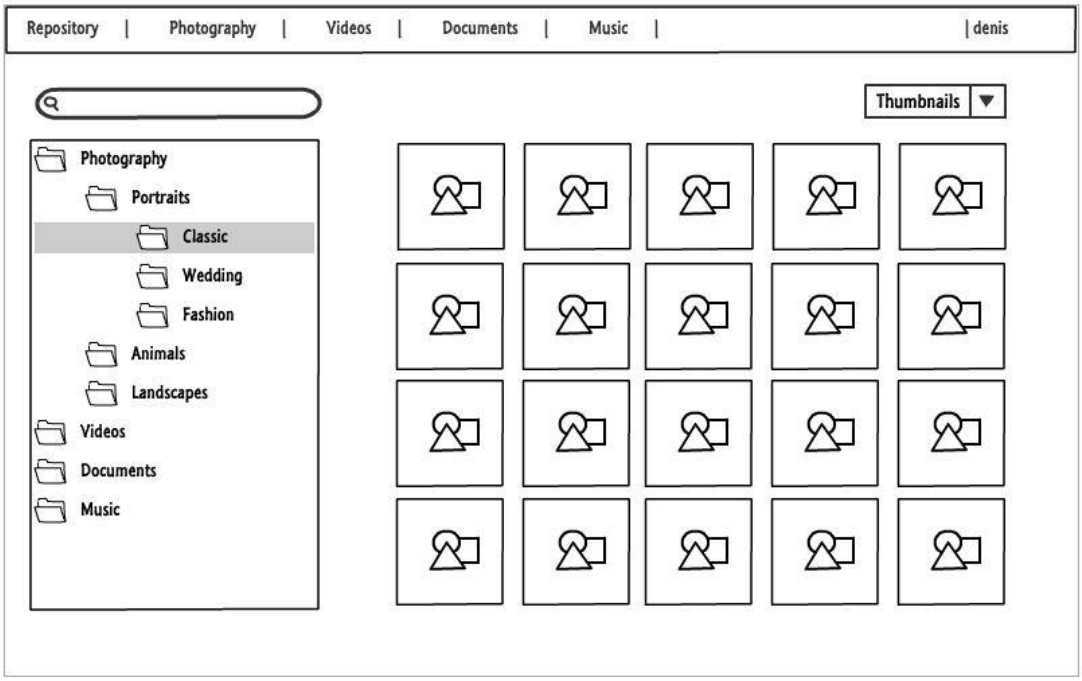

Figura 9 - *Mockup* da Galeria do *backoffice*

O segundo *mockup* (Figura 9) representa a janela principal do *backoffice* do utilizador, que contém a galeria do conteúdo, a árvore de nós e a caixa de pesquisa.

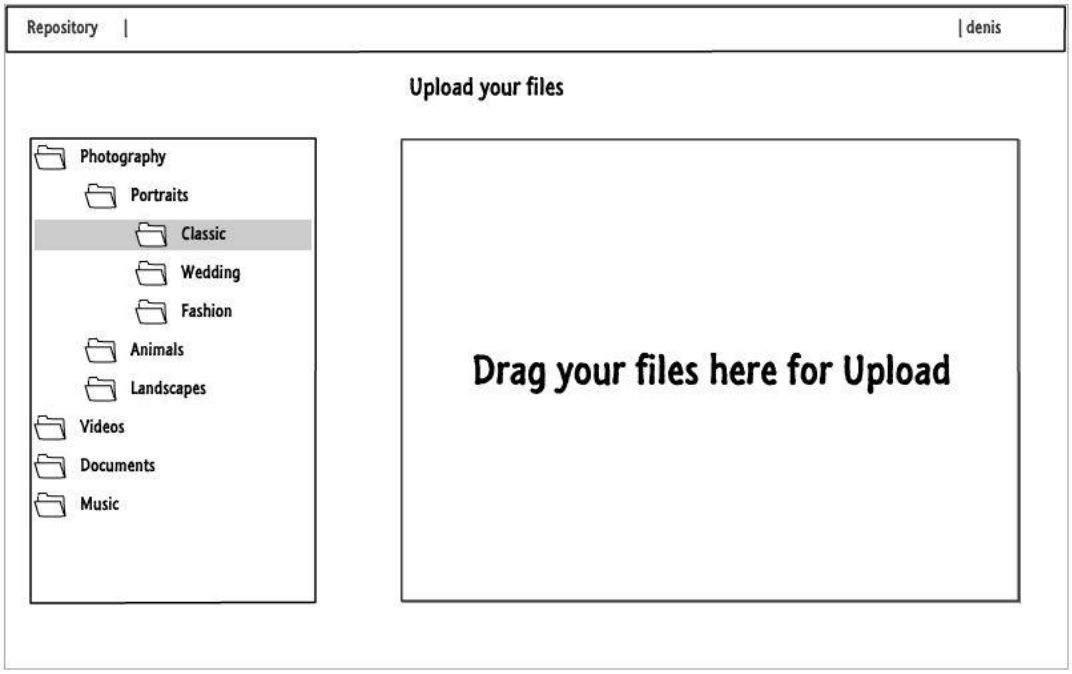

Figura 10 - *Mockup* da janela de *Upload* de conteúdo

O terceiro *mockup* (Figura 10) representa a janela de *upload* de conteúdo. Contém a árvore de nós para seleccionar o nó no qual é feito o *upload* e uma área onde o utilizador pode fazer *drag and drop* do conteúdo para fazer *upload*.

Figura 11 - *Mockup* da janela de visualização do conteúdo

O quarto *mockup* (Figura 11) representa a janela que mostra o conteúdo seleccionado. Contém um *container* onde é visualizado o conteúdo, uma área com botões de controlo e as informações sobre o conteúdo.

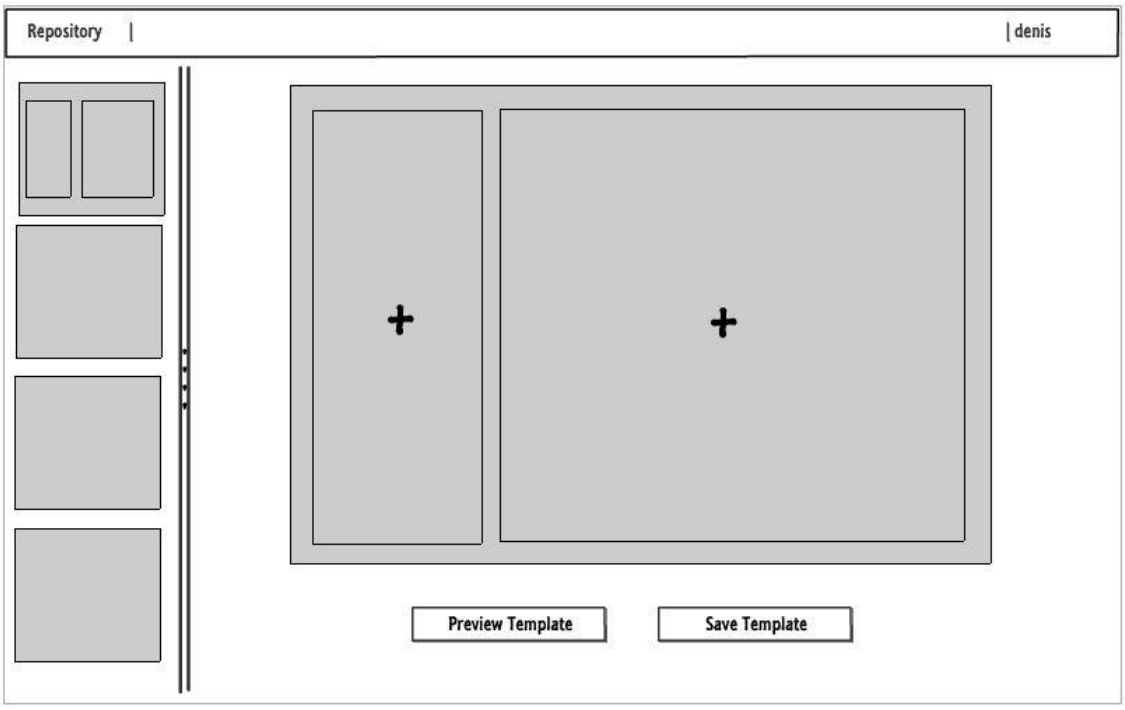

Figura 12 - *Mockup* da janela de personalização do template

Quinto *mockup* (Figura 12) representa a janela de *design* de *templates*. No lado esquerdo contém os modelos de *templates* disponíveis. Após clicar em algum *template*, aparece uma

previsualização desse no lado direito onde podemos adicionar os vários módulos à cada divisão do *template*. Por fim podemos pré-visualizar ou guardar o *template* personalizado.

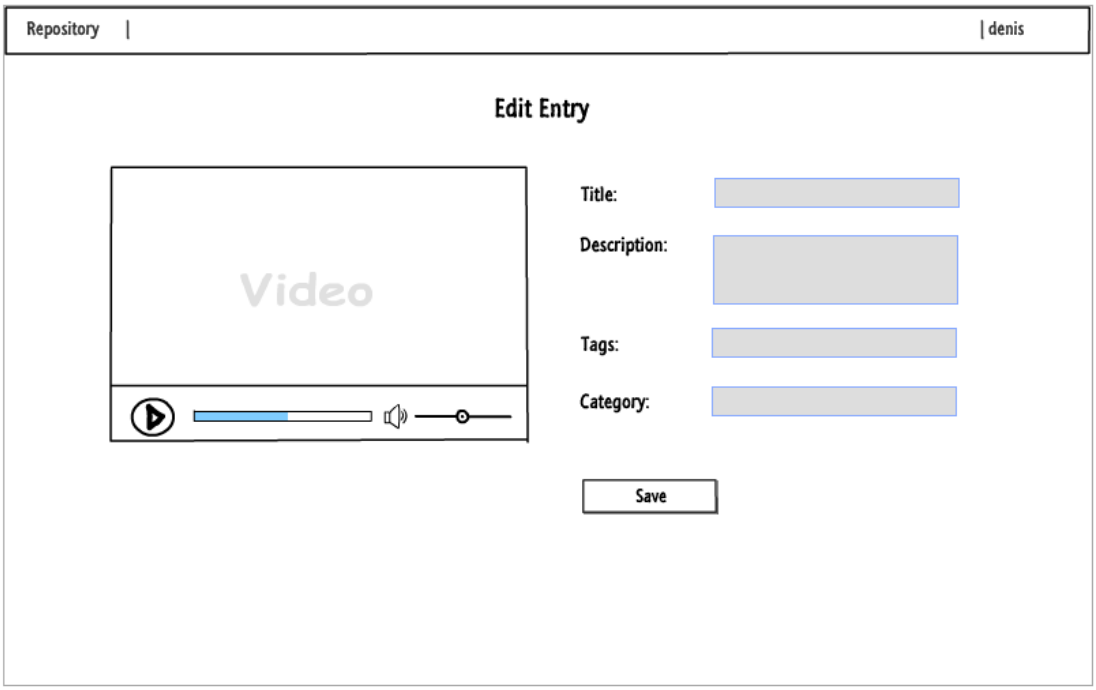

Figura 13 – *Mockup* da janela de edição de conteúdo

O sexto *mockup* representa a página onde o utilizador pode editar o conteúdo. No lado esquerdo contém uma visualização de conteúdo e no lado direito os campos que podem ser editados.

4 Arquitectura

4.1 Arquitectura Geral

Para entender melhor a arquitectura geral da aplicação é apresentado na Figura 14 um diagrama de componentes da aplicação. O diagrama foi simplificado para ser mais perceptível.

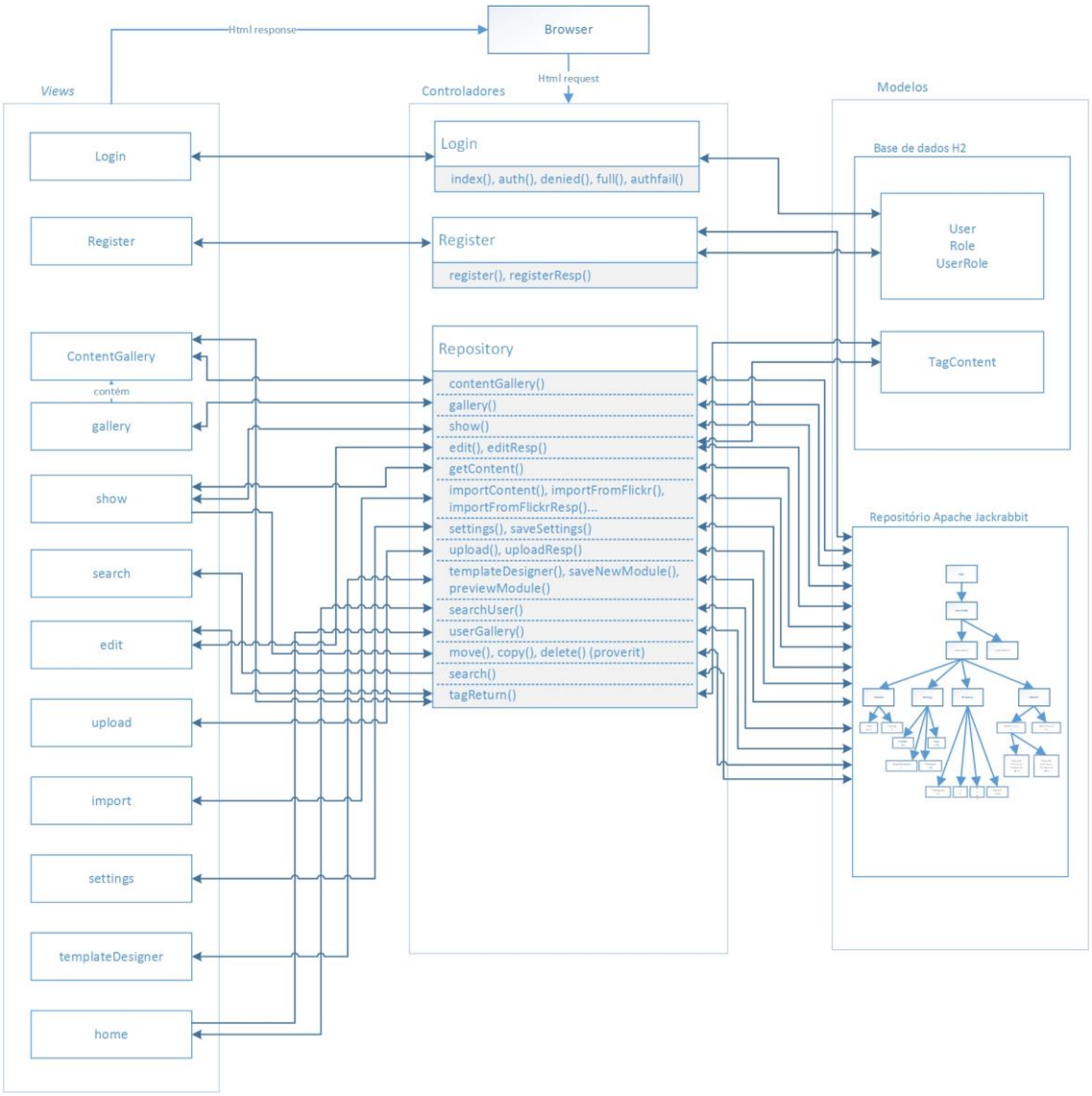

Figura 14 - Arquitectura geral da aplicação

Os principais blocos da aplicação são:

Browser de cliente – é um *browser* qualquer que suporta HTML. O *browser* funciona como um programa genérico com uma interface de utilizador. O utilizador da aplicação solicita as páginas Web ao servidor via *browser.* A interacção de utilizador com o sistema é realizada através de um *browser* via ligação *HyperText Transfer Protocol* (http), que é um protocolo de

interacção entre o cliente e o servidor. Sempre que existe alguma troca de informação entre cliente e servidor, é criada uma nova ligação independente.

Views – são as páginas *Groovy Server Pages* (GSP) com quais o utilizador interage e que são visualizados no *browser*.

Controladores – neste bloco estão incluídos os controladores e os métodos correspondentes.

Modelos – representam os dados. Como os dados principalmente estão armazenados no repositório Apache Jackrabbit, a utilização da base de dados é mínima e serve apenas para armazenar os dados de acesso de utilizadores e os *Tags*.

4.2 Controladores

Os controladores e os métodos existentes, junto com uma pequena descrição para facilitar o entendimento de papel de cada, encontram-se listados nas tabelas abaixo.

LoginController	**Controlador criado pelo *plugin* Spring Security. É responsável pela correcta autenticação dos utilizadores na aplicação**
index()	Redirecciona para a página principal de galeria de utilizador se ele está autenticado ou para o método auth() se não está autenticado
auth()	Mostra a página de autenticação de utilizador
denied()	Mostra a página correspondente ao acesso negado
full()	A página de autenticação para os utilizadores com opção de *cookie* "*Remember me*" activada
authfail()	Redirecciona para a página de autenticação de utilizador mostrando a mensagem de aviso no caso de falhar a autenticação

Tabela 2 - Descrição de controlador LoginController

LogoutController	**Outro controlador que foi criado pelo *plugin* Spring Security. Tem como responsabilidade a correcta saída de utilizadores da aplicação**
índex()	Redirecciona para página inicial da aplicação e faz *logout* da sessão de repositório de Jackrabbit

Tabela 3 - Descrição de controlador LogoutController

RegisterController	**Controlador que é responsável pelo registo de novos utilizadores na aplicação**
register()	Mostra a página de registo de utilizador
registerResp()	Regista o novo utilizador na aplicação e cria os nós necessários no repositório de Jackrabbit para ele

Tabela 4 - Descrição de controlador RegisterController

RepositoryController	Controlador que é responsável pelas todas as restantes funcionalidades da aplicação
addToLocal()	Transfere o conteúdo binário de ficheiro que encontra-se *online* para o repositório
authorize_[service]()	Faz autorização de utilizador no serviço indicado como [service]
changeAccess()	Muda o acesso de conteúdo de privado para público ou vice-versa
contentGallery()	Mostra a página contentGallery.gsp
deleteEntryResp()	Remove o conteúdo seleccionado
deleteNode()	Remove o nó da árvore
edit()	Mostra a página edit.gsp
editResp()	Edita o conteúdo seleccionado
getContent()	Faz render dos dados binários do conteúdo seleccionado conforme o tipo de ficheiro
importContent()	Mostra a página importContent.gsp
importFrom[Service]()	Mostra a página importFrom[Service].gsp, onde [service] é nome de um dos serviços
importFrom[Service]Resp()	Guarda no repositório o conteúdo do serviço seleccionado pelo utilizador
moveEntry()	Move o conteúdo para o outro nó da árvore hierárquica
renameNode()	Muda o nome do nó da árvore
saveNewModule()	Guarda no repositório o trecho de código criado pelo utilizador
saveSettings()	Guarda as definições
saveUserTemplate()	Guarda no repositório as definições de *template* da galeria pública configuradas pelo utilizador
search()	Retorna a lista de conteúdo que corresponde aos parâmetros da pesquisa
searchUser()	Verifica a existência de utilizador
settings()	Mostra a página settings.gsp
show()	Mostra o conteúdo seleccionado e a informação correspondente
tagreturn()	Retorna uma lista de *Tags* que contêm os caracteres recebidos em formato *JavaScript Object Notation* (JSON)
templateDesigner()	Mostra a página templateDesigner.gsp
upload()	Mostra a página upload.gsp
upload_resp()	Guarda no repositório o ficheiro recebido e a informação extraída usando Apache Tika
userGallery()	Faz render do *template* escolhido da galeria pública do utilizador juntando os módulos configurados
contentGallery()	Mostra a página contentGallery.gsp
templatePreview()	Mostra o esqueleto do *template* seleccionado na página templateDesigner.gsp
download()	Descarrega o ficheiro seleccionado para o computador
saveUserInfo()	Guarda as informações pessoais do utilizador
savePubGalSettings()	Guarda as definições da galeria pública do utilizador

Tabela 5 - Descrição de controlador RepositoryController

4.3 Models

Nas tabelas abaixo são listados os modelos de dados existentes na aplicação.

User	**Classe de domínio de utilizador**
String username	Username do utilizador
String password	Palavra-chave do utilizador
boolean enabled	Variável para definir se o utilizador está activado
boolean accountExpired	Variável para definir se a conta do utilizador está expirada
boolean accountLocked	Variável para definir se a conta do utilizador está bloqueada
boolean passwordExpired	Variável para definir se a palavra-chave do utilizador está expirada

Tabela 6 - Descrição de classe de domínio de Utilizador

Role	**Classe de domínio de papel de utilizador**
String authority	Papel de utilizador

Tabela 7 - Descrição de classe de domínio de papel de utilizador

UserRole	**Classe de domínio para mapeamento das classes User e Role com multiplicidade de muitos para muitos**
User user	Instância de classe de domínio User
Role role	Instância de classe de domínio Role

Tabela 8 - Descrição de classe de domínio de mapeamento das classes User e Role

TagContent	**Class de domínio de tags de conteúdo**
String tag	Tag de conteúdo
int cnt	Contador de utilizações de *tag*

Tabela 9 - Descrição de classe de domínio de *tags* de conteúdo

4.4 Views

Nas tabelas abaixo são listados os *Views* existentes na aplicação.

índex.gsp	Página inicial da aplicação. Contém o campo para de pesquisa de galeria pública do utilizador.
contentGallery.gsp	Página principal de galeria do utilizador. Contém duas divisões como as principais partes, numa é carregada a árvore hierárquica de nós e noutra é carregada a página gallery.gsp
edit.gsp	Página para a edição de conteúdo. Contém os campos título,

	descrição, *tags* e categorias que podem ser editados
importContent .gsp	Página com as várias opções de serviços de onde importar o conteúdo
gallery.gsp	Página que contém os *thumbnails* de conteúdo
search.gsp	Página que contém o conteúdo que é retornado pelo método da pesquisa
settings.gsp	Página com as definições da aplicação do utilizador
show.gsp	Página para visualização de conteúdo seleccionado
templateDesigner.gsp	Pagina onde utilizador pode escolher o *template* para sua galeria pública
upload.gsp	Página para *upload* de conteúdo para o repositório

Tabela 10 - Descrição dos Views da aplicação

4.5 Arquitectura do repositório Apache Jackrabbit.

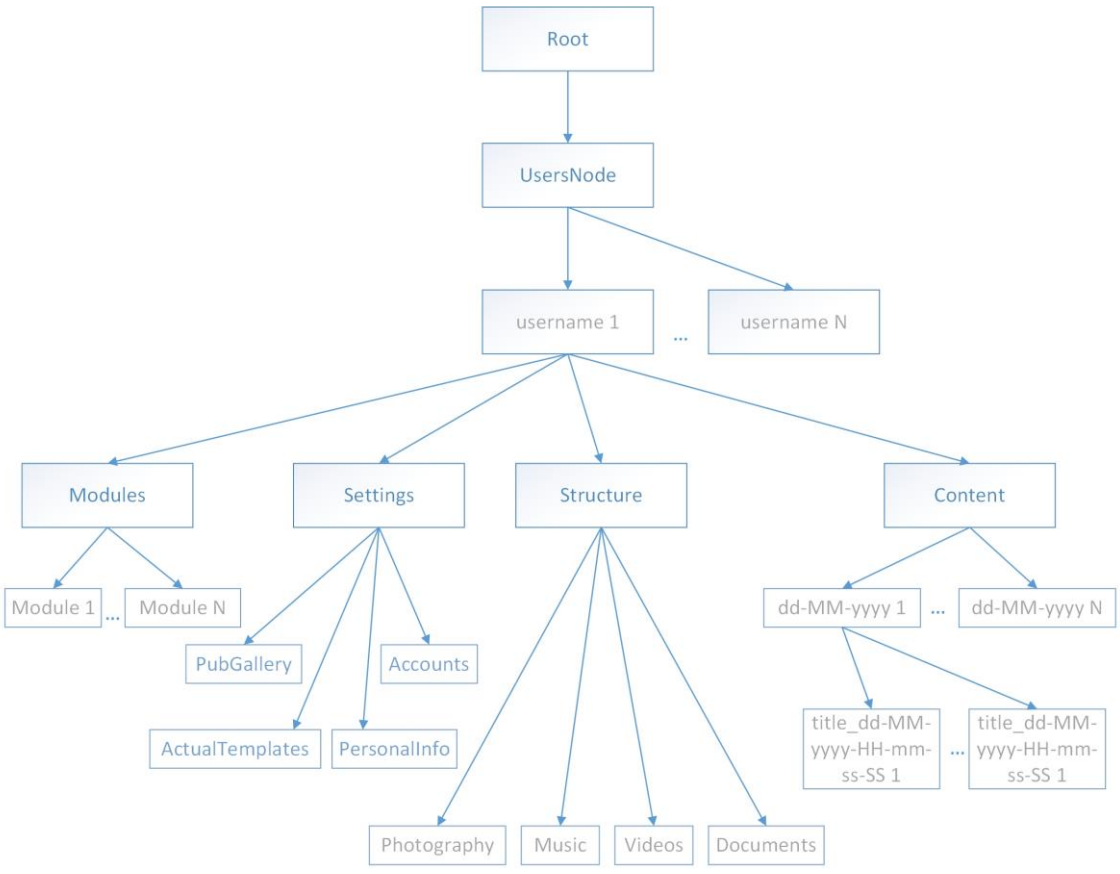

Figura 15 - Arquitectura do repositório Apache Jackrabbit

Os conteúdos e os dados correspondentes são armazenados no repositório Apache Jackrabbit que se encontra no servidor.

Os nós do repositório podem ser imaginados como um sistema de ficheiros do sistema operativo para maior conveniência do utilizador. Assim existem os nós-pastas que contêm os

outros nós-pastas e os nós-ficheiros que podem conter os dados binários, os metadados e outras informações correspondentes.

No topo da hierarquia encontra-se o nó raiz do repositório, chamado *Root*. Este nó é único para toda a aplicação. Todos os restantes nós estão armazenados dentro do *Root*.

Por baixo do nó raiz situa-se o nó chamado *UsersNode* que inclui as pastas dos utilizadores registados na aplicação. Este nó também é um nó único para toda a aplicação.

O nó *UsersNode* contém os nós- pastas dos utilizadores. Para cada utilizador registado é criado um nó-pasta com o nome igual ao *username* do utilizador. Este nó contém toda a informação, o conteúdo e as definições do utilizador.

Dentro de cada nó do utilizador encontram-se 4 nós-pastas: *Modules*, *Settings*, *Structure* e *Content*.

No nó *Modules* estão armazenados os módulos criados pelo utilizador.

O nó *Settings* armazena as definições da aplicação do utilizador. É neste nó que encontram-se armazenados os dados pessoais do utilizador (nome, email, sexo, etc), as definições da galeria pública e os dados de autorização dos serviços externos.

O nó *Structure* contém a estrutura hierárquica do conteúdo do utilizador. É esta estrutura que está associada à árvore hierárquica de nós da aplicação Web.

No nó *Content* está armazenado todo o conteúdo do utilizador, incluindo os ficheiros binários, os metadados e as outras informações correspondentes.

4.6 Gestão de Segurança e Autenticação de Utilizadores

4.6.1 Framework Spring Security

Framework Spring Security [22] é um *framework* de autentificação e controlo de acesso poderoso e personalizável. Uma das maiores vantagens do Spring Security é flexibilidade, o que fez esse *framework* tão popular para garantir a segurança nas aplicações de Spring. O projecto foi iniciado em 2003 com o nome Acegi Security. Posteriormente foi incorporado para Spring. Pela primeira vez foi apresentado com o nome Spring Security 2.0.0 em Abril de 2008.

4.6.2 Plugin Spring Security

Para a gestão de segurança da aplicação e autentificação de utilizadores foi utilizado *plugin* Spring Security [23]. Este *plugin* simplifica a integração de *framework* Spring Security nas aplicações de Grails. O *plugin* fornece muitas opções de configuração e personalização.

Para incluir o *plugin* no projecto, basta incluí-lo nas dependências do projecto e correr o *script* que automaticamente gera os modelos de dados básicos para guardar a informação dos utilizadores e os controladores para autenticação.

Foram então criados 3 classes de domínio: User, Role e UserRole e 2 controladores LoginController e LogoutController.

4.6.3 Autenticação

A correcta autenticação dos utilizadores na aplicação é assegurada pelo Spring Security. Para autenticar-se na aplicação é necessário introduzir o *username* e a palavra-chave nos campos correspondentes. Ao submeter os dados de autenticação, o acesso é validado na base de dados. Após a autenticação com sucesso o utilizador é redireccionado para a página principal da sua galeria, no caso contrário é apresentada a mensagem de erro e é pedido *login* e a palavra-chave novamente.

4.6.4 Gestão de Segurança

Para limitar o acesso a alguns métodos do controlador ou páginas são usados anotações *@Secured* fornecidas pelo *plugin*. Para definir quem pode aceder a certo controlador, método ou página, basta passar a lista de regras básicas antes da classe ou antes do método correspondente.

```
class RepositoryController {

    @Secured(['permitAll'])
    def returnEntry(){

    }

    @Secured(['ROLE_USER'])
    def uploadEntry(){

    }
```

Figura 16 - Utilização de anotações

A figura 16 mostra um exemplo de utilização das anotações. A anotação *@Secured([‘permitAll’])* significa que todos os utilizadores têm acesso ao método returnEntry(). Já a anotação *@Secured([‘ROLE_USER’])* significa que apenas os utilizadores com role “USER” têm acesso ao método uploadEntry().

4.7 Interacção com os serviços existentes

Para a integração completa das aplicações externas, é necessário fazer autenticação nelas. Para esse fim é usada especificação OAuth [34]. O processo de autenticação usando esta especificação é semelhante em todos os casos usados neste trabalho. Nos parágrafos seguintes é descrita essa especificação e o processo de autenticação de um utilizador no Flickr, como exemplo.

4.7.1 OAuth

OAuth é um protocolo aberto, simples e seguro, que permite a autenticação e interacção de utilizadores com os serviços externos usando o seu nome. A informação de utilizador é transferida em segurança sem revelar a sua identidade. Nos serviços integrados oAuth é usado para verificar se a aplicação que está a tentar interagir com o serviço em nome do utilizador tem as permissões que o utilizador lhe concedeu.

4.7.2 Autenticação no serviço externo (Flickr)

Para começar a usar a API do Flickr, em primeiro lugar é necessário registar a aplicação no *website* do Flickr e obter a *API Key* e a *API Secret*. Tendo a *API Key* e a *API Secret*, o fluxo de autenticação usando OAuth resume-se em três passos.

1. O primeiro passo é obter a *Request Token* usando *a API Key*. Este é um *token* temporário que será usado para autenticar o utilizador na aplicação. Após fazer a chamada são retornados dois parâmetros, *oauth_token* que é *Request Token* e *oauth_token_secret* que é *Request Token Secret*.

2. O passo seguinte é redireccionar o utilizador para a página de autorização do Flickr, onde são requeridas as permissões para realização das operações. Depois de completar a autorização, o Flickr redirecciona o utilizador de volta para a aplicação usando o endereço de *callback* especificado passando o *oauth_verifier*.

3. O terceiro e último passo, é trocar o *Request Token* por um *Access Token*, que deve ser armazenado pela aplicação e usado para fazer os pedidos autorizados ao Flickr.

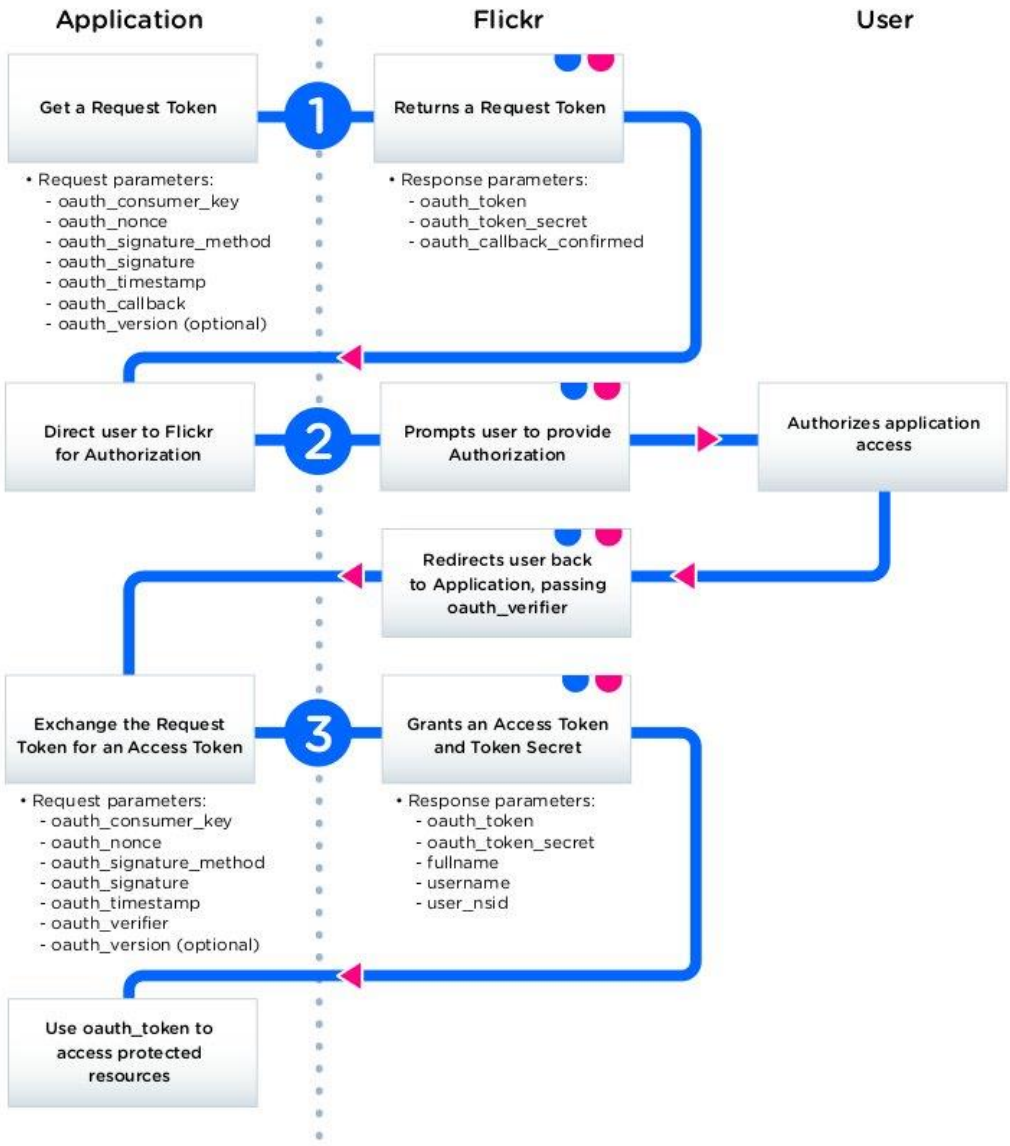

Figura 17 - Esquema de autenticação no Flickr usando oAuth

4.8 Funcionalidades

4.8.1 Registo de utilizadores novos

Para começar a ter o repositório pessoal, é necessário registar-se na aplicação. Para isso na página register.gsp, é preciso introduzir os dados de registo. A seguir os dados são enviados para o método registerResp() do controlador RegisterController, que verifica se existe algum utilizador na base de dados com o mesmo *username*. No caso afirmativo, é mostrada a mensagem de erro a indicar que o utilizador com o *username* introduzido já existe. No caso de não existir nenhum utilizador com o *username* correspondente, o utilizador é guardado na base de dados e é criada toda a estrutura de nós necessária para o repositório pessoal do utilizador no repositório Apache Jackrabbit.

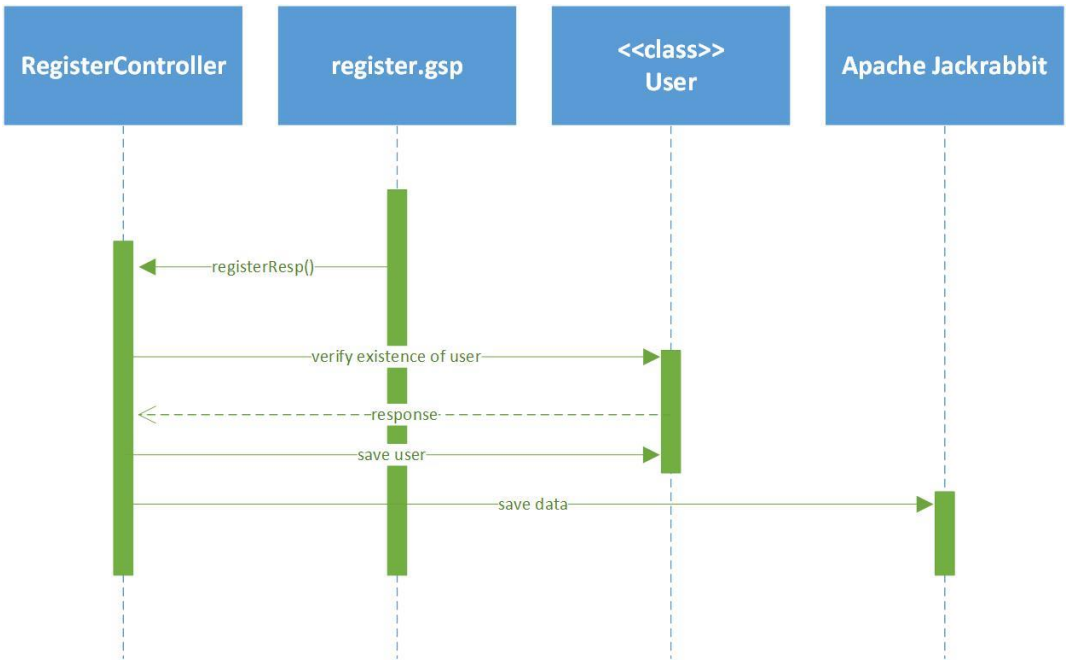

Figura 18 - Diagrama de interacção de registo de utilizadores novos

4.8.2 Preenchimento da árvore hierárquica de nós

Uma grande vantagem da aplicação desenvolvida é a existência da árvore hierárquica que permite organizar o conteúdo do utilizador de maneira mais conveniente às suas necessidades. Para preenchimento da árvore com os nós existentes é utilizada a tecnologia *Asynchronous Javascript and XML* (AJAX). Na página é feita uma chamada para os métodos getTreeParent() e getTreeChild() do controlador RepositoryController através de script. Esses métodos retornam uma lista de nós em formato JSON de primeiro nível e os restantes nós correspondentes com quais é preenchida a árvore.

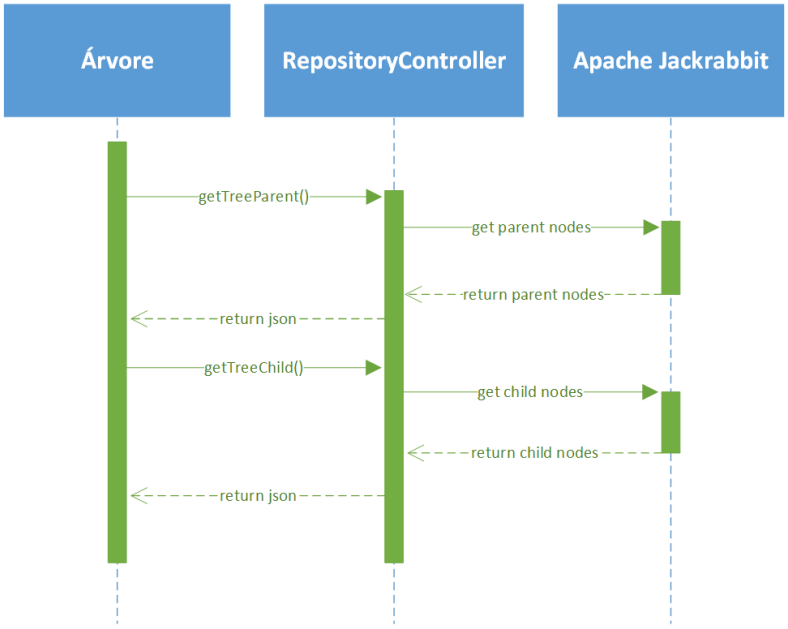

Figura 19 - Diagrama de interacção de preenchimento da árvore hierárquica de nós

4.8.3 Visualização de galeria de conteúdo

É a funcionalidade que é responsável pela visualização de conteúdo seleccionado. Ao clicar num dos nós da árvore é chamado o método gallery() do controlador RepositoryController passando o nó seleccionado como parâmetro. O método gallery() executa uma *query* no repositório Jackrabbit e retorna uma lista de conteúdos correspondentes ao nó seleccionado. A seguir é carregada a página galery.gsp com a lista de conteúdo seleccionado. Nesta página para cada conteúdo é feita a chamada para o método returnThumb() para mostrar o *thumbnail* correspondente.

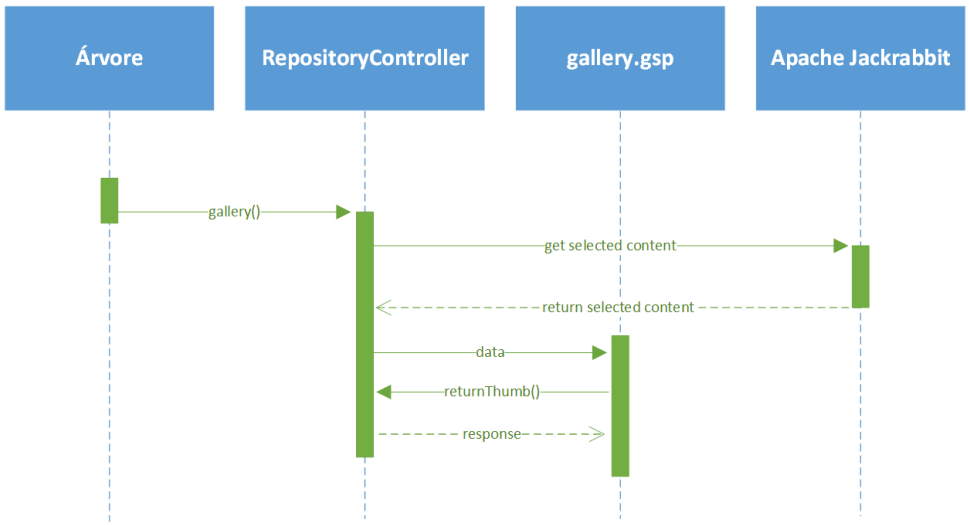

Figura 20 - Diagrama de interacção de visualização de galeria de conteúdo

4.8.4 Visualização de conteúdo seleccionado

Para visualizar o conteúdo desejado e a informação correspondente é chamado o método show() que recebe como parâmetro o conteúdo seleccionado e faz um pedido ao repositório de Jackrabbit para obter a informação do conteúdo. Depois disso é carregada a página show.gsp com a informação recebida. Para visualizar o conteúdo seleccionado é chamado o método getContent() que faz um pedido ao repositório Jackrabbit e recebe os dados binários, dos quais por fim faz a visualização, dependendo do formato de ficheiro.

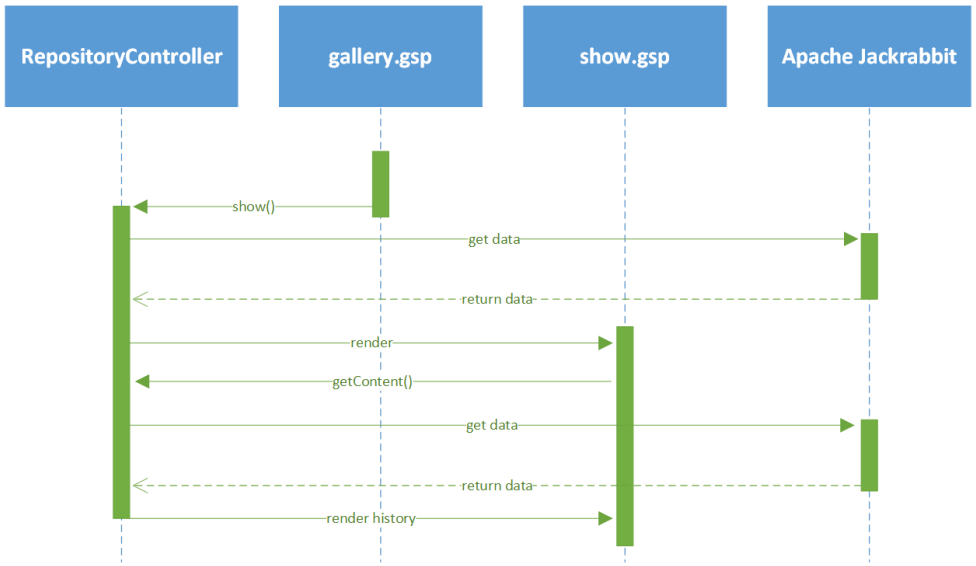

Figura 21 - Diagrama de interacção de visualização de conteúdo seleccionado

4.8.5 Carregar conteúdo

Quando o conteúdo é carregado na página, ele é enviado junto com a categoria de destino que foi seleccionada para o método uploadResp() , que extrai toda a informação sobre o ficheiro e guarda-o no repositório junto com essa informação. Se o ficheiro foi guardado com sucesso é enviada uma mensagem com "status: 200".

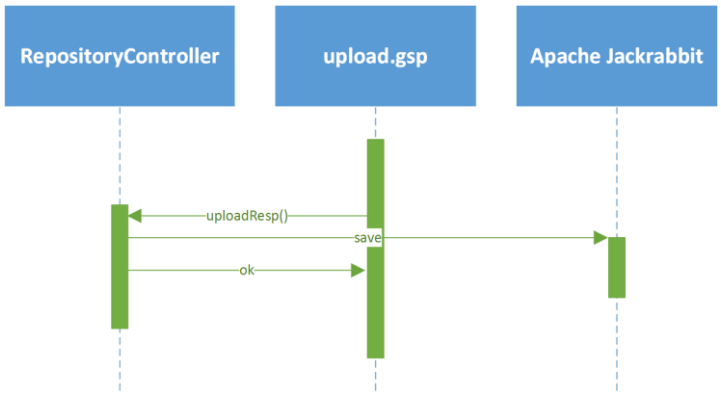

Figura 22 - Diagrama de interacção de *upload* de conteúdo

4.8.6 Edição de conteúdo

A aplicação permite editar o conteúdo seleccionado. Na página edit.gsp, que é responsável pela essa função, é possível editar os *tags*, o título, a descrição, a as categorias do conteúdo seleccionado. No campo de *tags*, ao introduzir os caracteres, é feita a chamada para o método tagReturn() que retorna os *tags* que contém os caracteres introduzidos utilizando a tecnologia AJAX. Ao submeter a informação, ela é enviada para o método editResp() que guarda-a no

repositório. Por fim o método redirecciona o utilizador para a janela principal da galeria de conteúdo.

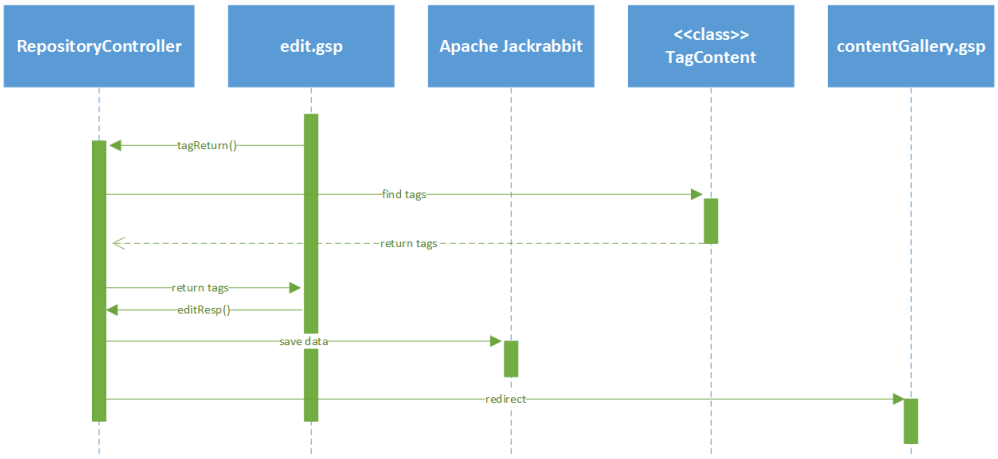

Figura 23 - Diagrama de interacção de edição de conteúdo

4.8.7 Pesquisar conteúdo

Ao submeter o termo da pesquisa ele é enviado para o método search(). A seguir o método executa uma *query* no repositório que por sua vez retorna uma lista de conteúdo que corresponde aos parâmetros da pesquisa. No fim é carregada a página search.gsp com o conteúdo retornado pelo método.

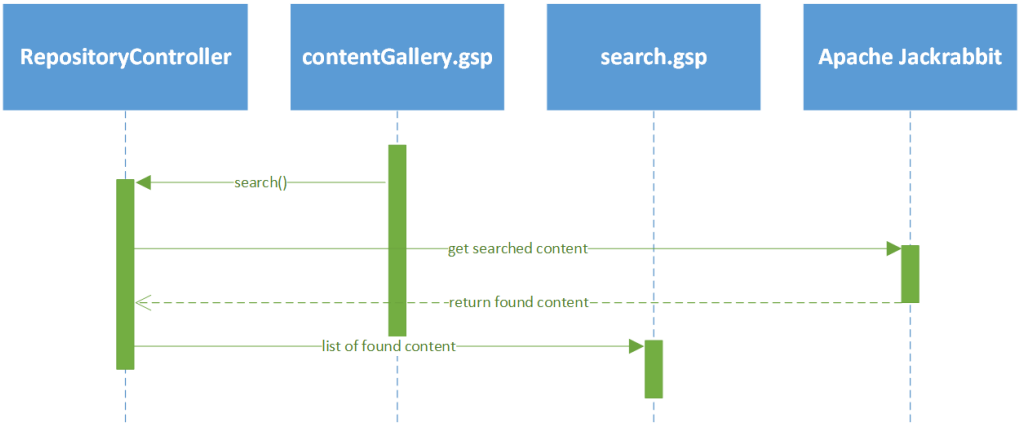

Figura 24 - Diagrama de interacção de pesquisa de conteúdo

4.8.8 Importar conteúdo

Para importar o conteúdo de outros serviços existentes, é preciso na página de import.gsp escolher um dos serviços disponíveis. Na figura 25 está exemplificado o caso de Flickr. Depois de seleccionar o serviço pretendido, é aberta uma janela importFromFlickr.gsp que faz a chamada de método importFromFlickr(). No método é feito um pedido à API de Flickr, que retorna a lista de conteúdo existente. A seguir o método envia essa lista para a página importFromFlickr.gsp onde o utilizador pode escolher o conteúdo desejado. Depois de seleccionar e submeter o conteúdo, é chamado o método importFromFlickrResp() que guarda o conteúdo recebido no repositório.

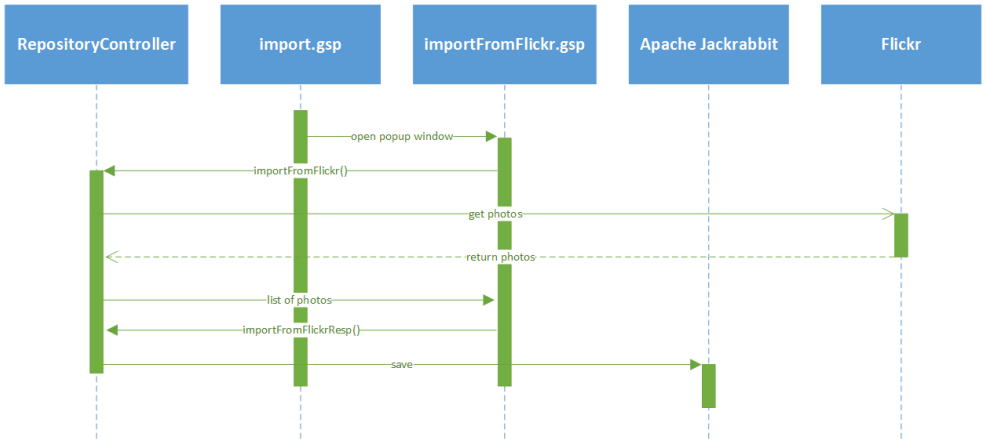

Figura 25 - Diagrama de interacção de *import* de conteúdo

4.8.9 *Tags*

Os *Tags* estão armazenados na base de dados do Grails. A classe de *Tag* consiste em dois campos "*String tag*" correspondente ao *tag* e "*int cnt*" correspondente ao contador de utilização de *tag*. Quando algum utilizador agrega algum tag ao conteúdo, o contador é incrementado.

A aplicação usa a tecnologia AJAX para fazer pedidos ao servidor para propostas de *tags* mais utilizados. Para isso, quando o utilizador introduz os caracteres no campo, esses caracteres são enviados para o servidor, onde é chamado o método tagReturn(). Este método executa uma *query* que retorna uma lista de *tags* que contêm os caracteres recebidos. Por fim essa lista é retornada à página e os *tags* são apresentados ao utilizador pela ordem de número de utilizações.

4.8.10 *Template Designer*

Uma das funcionalidades importantes da aplicação consiste no construção da galeria pública personalizada aplicando vários *templates,* semelhante aos CMS. Para isso na página templateDesigner.gsp o utilizador tem escolher um dos *templates* existentes, uma acção que passa o *template* escolhido como o parâmetro para o método templatePreview() que faz *render* do esqueleto do *template* na página. Após ter o esqueleto do template na página o utilizador precisa de escolher um módulo para cada área de *template* escolhido.

O utilizador também tem a possibilidade de criar um novo módulo. Neste caso depois de escrever o código do módulo e guardar o módulo desenvolvido, é chamado o método saveNewModule() do controlador RepositoryController(), que cria um novo nó no repositório de Jackrabbit dentro do nó Modules.

Por fim, quando todas as áreas do *template* são preenchidas com módulos e o utilizador guarda o *template* criado, é chamado o método saveUserTemplate() que guarda o id do *template* escolhido e os módulos definidos no repositório.

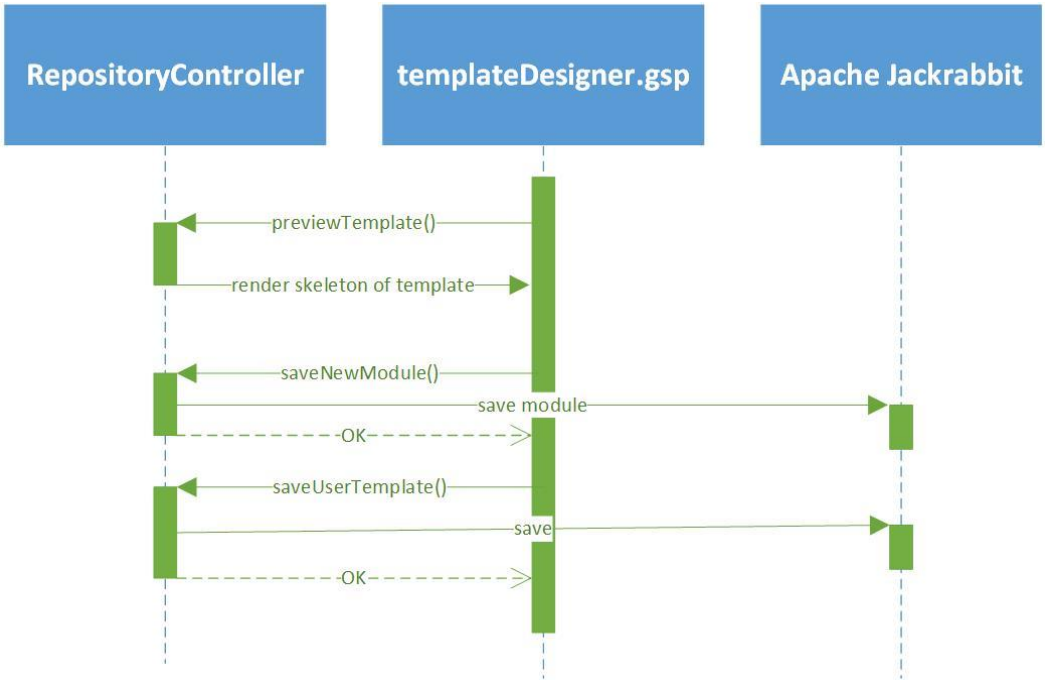

Figura 26 - Diagrama de interacção de *Template Designer*

4.8.11 Galeria Pública

Para aceder à galeria pública do utilizador basta escrever o nome do utilizador no campo de pesquisa na página inicial. Para verificar se o utilizador pesquisado existe e tem a sua galeria activada, o nome introduzido é enviado para o método searchUser(), que faz a pesquisa na base de dados e faz as verificações.

No caso, se o utilizador existe e tem a sua galeria pública activada é chamado o método userGallery() que faz *render* do *template* definido. Dentro do *template* para cada secção é chamada a função renderModule() que é responsável pela *render* do módulo correspondente.

Deste modo a página de galeria pública é construída "*on the fly*" dependendo do *template* e dos módulos definidos.

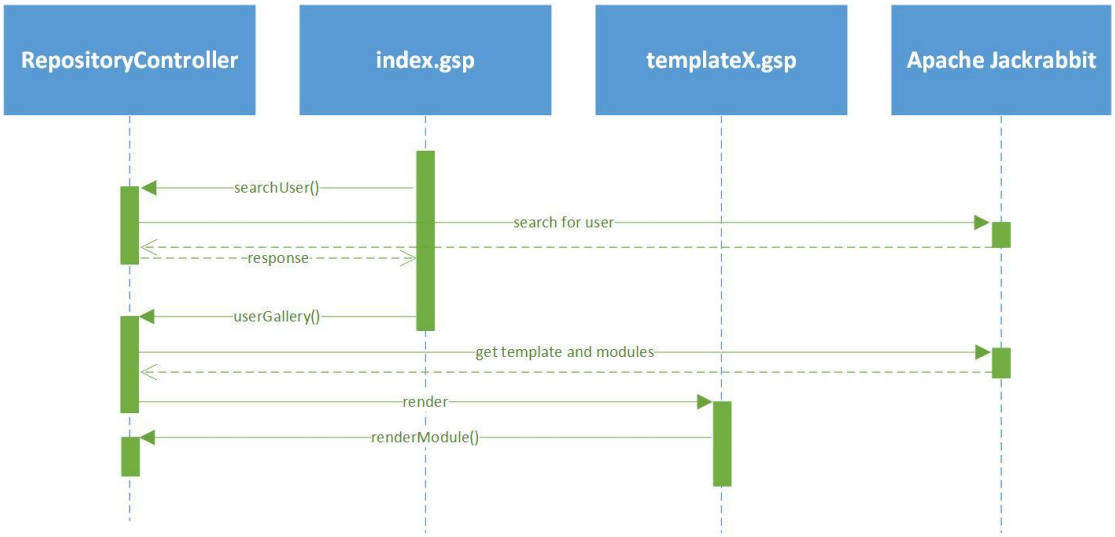

Figura 27 - Diagrama de interacção da Galeria Pública

4.9 Implementação da Interface

4.9.1 Front-End Framework (Zurb Foundation)

Para construir a maquete do *website* foi utilizado o *framework* de código aberto Foundation 5 [24] da empresa norte-americana Zurb. Este *framework* serve para desenvolvimento rápido de *websites* adaptativos. Foundation contém tipografia, botões, formas, componentes de navegação e outros elementos de interface baseados em HTML e CSS.

A maior vantagem de Zurb Foundation 5 é suporte de *design* adaptativo, o que permite as páginas web a adaptar-se dinamicamente à resolução do ecrã. Assim a aplicação consegue adaptar-se às dimensões correspondentes da janela do *browser* quer seja no ecrã de *smartphone* ou num ecrã de tamanho maior.

Grelha

Para criar um *layout* flexível e adaptativo para qualquer dispositivo, foi utilizado o sistema de grelha fornecido pelo Zurb Foundation, que consiste numa grelha com 12 colunas de tamanhos iguais que consegue adaptar-se aos tamanhos arbitrários.

O trecho de código da figura 28 exemplifica a utilização de sistema de grelhas.

```
<div class="row">
  <div class="small-2 large-4 columns">...</div>
  <div class="small-4 large-4 columns">...</div>
  <div class="small-6 large-4 columns">...</div>
</div>
```

Figura 28 - Utilização de grelha de Zurb Foundation

É criada uma divisão-linha que está dividida em 3 colunas com os tamanhos definidos para ecrãs grandes e ecrãs pequenos. Para os ecrãs grandes a largura de colunas está definida como 4 colunas de grelha cada. Já para os ecrãs pequenos a largura de cada coluna está definida como 2,4, e 6 colunas da grelha correspondente. Nas figuras 29 e 30 é ilustrado o resultado no ecrã de portátil e no ecrã de *smartphone*.

Figura 29 - Resultado de grelha no portátil

Figura 30 - Resultado de grelha no *smartphone*

Barra de topo

Para navegação na aplicação é utilizado *Top Bar*, é uma funcionalidade que se trata de uma barra - cabeçalho da aplicação, que também é flexível e consegue adaptar-se ao qualquer tamanho do ecrã. Este elemento está contido em todas as páginas da aplicação.

Quando o utilizador não está autenticado na aplicação, a barra tem no seu lado direito os botões de registo e de autenticação, e no lado esquerdo o nome da aplicação (Figura 31).

Figura 31 - Aparência de barra de topo para utilizador não autenticado

Ao autenticar-se na aplicação a barra começa a ter no lado direito um botão com o *username* do utilizador. Ao colocar o cursor do rato no botão, aparece uma lista do tipo *dropdown* com as várias opções de operações como *Upload* e *Import* de Conteúdo, Definições, *Design* de *Templates* e *Logout* (Figura 32).

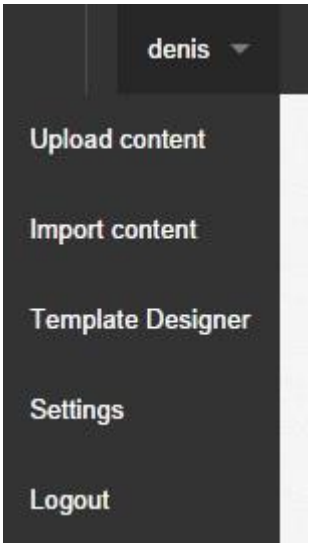

Figura 32 - Dropdown

No lado esquerdo o nome da aplicação permanece igual, mas já contém a hiperligação para a página principal da galeria do utilizador. A seguir do nome da aplicação, ficam os botões com as categorias do primeiro nível da hierarquia de repositório para uma navegação rápida e fácil (Figura 33).

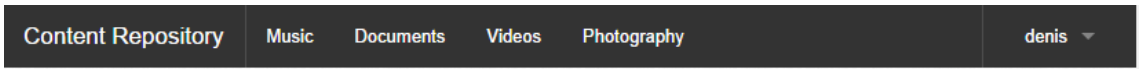

Figura 33 - Aparência de barra de topo para utilizador autenticado

4.9.2 Árvore hierarquica de nós (jsTree)

jsTree [25] é um *plugin* de jQuery de código aberto que permite a criação de arvores de nós interactivas. Este *plugin* é facilmente extensível e configurável. Suporta HTML e JSON como fonte de dados e permite carregamento usando AJAX. Para preencher a árvore é enviado um pedido ao servidor que responde com a lista de nós em formato JSON.

4.9.3 Utilização de Tags (Select2)

Select2 [26] é uma alternativa a *select-boxes* de HTML. É baseado no jQuery e suporta pesquisas, visualização de dados remotos, ordenação e *scrolling* infinito de dados.

4.9.4 Upload de ficheiros (DropZone)

Dropzone.js [27] é um *plugin* de jQuery de código aberto que permite carregar os ficheiros para a página web usando a função de HTML5 *Drag-and-drop*. Este *plugin* suporta a previsualização de ficheiros antes de carrega-los para o servidor.

4.9.5 Galeria pública (YoxView)

YoxView [28] é um *plugin* de jQuery para criação de galerias de conteúdo multimédia do tipo *LightBox*. Este *plugin* suporta a visualização de vários formatos de ficheiros, *slideshow* e navegação usando o teclado.

4.9.6 Editor de HTML (CKEditor)

Para facilitar a construção de módulos próprios em HTML, foi utilizado editor de texto HTML de código aberto CKEditor [29]. É um editor de tipo What You See Is What You Get (WYSIWYG) que oferece a possibilidade de criar o conteúdo web na própria página da aplicação em que está usado.

5. Resultados

Neste capítulo é apresentada a aplicação desenvolvida ao longo deste trabalho. Será feita a comparação com as aplicações estudadas no capítulo 2. Também vai ser demonstrado o resultado final obtido e as interfaces da aplicação, onde será descrita e exemplificada a utilização do sistema implementado.

5.1 Página inicial

Ao entrar na aplicação é apresentada ao utilizador uma página com o nome da aplicação e um campo de entrada (Figura 34). Para aceder à página de galeria pública de utilizador basta introduzir o *username* no campo correspondente e pressionar a tecla *ENTER*. De seguida, se o utilizador existe e tem a sua galeria pública activada, a página redirecciona para a galeria do utilizador (Figura 34). Se o utilizador introduzido não é encontrado ou tem a sua galeria pública desactivada, aparece uma mensagem de erro correspondente.

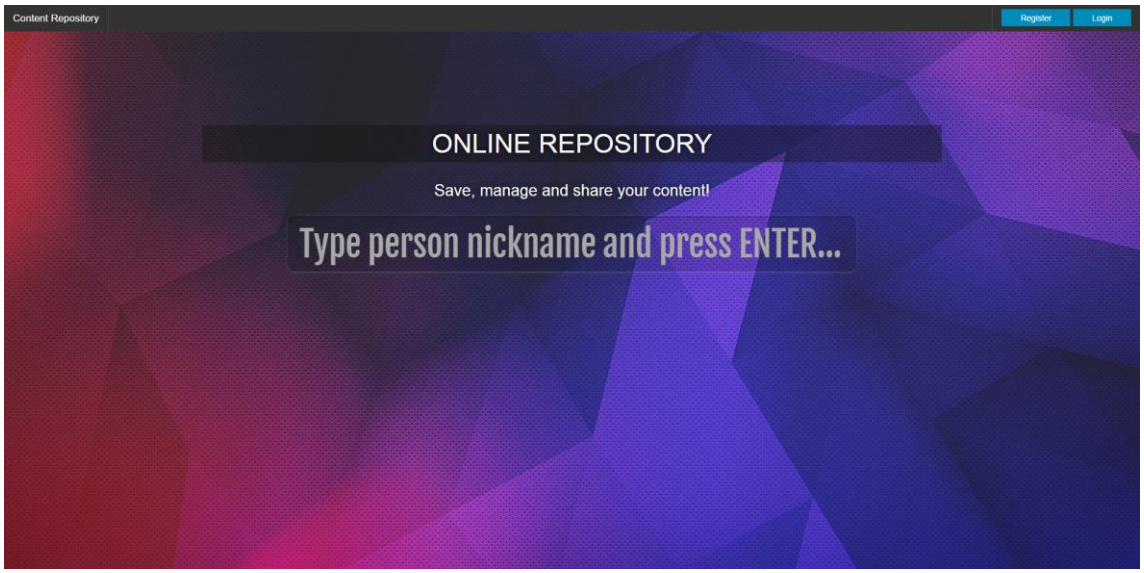

Figura 34 - Página Inicial

5.2 Página de *Login*

A autenticação na aplicação é feita a partir de um formulário que é composto por dois campos, um para *username* e outro para a palavra-chave. Existe também uma *checkbox* com a opção de *Remember Me*, que permite memorizar a identificação de utilizador entre as sessões utilizando os *cookies* do *browser*. Deste modo, se o utilizador seleccionar essa opção ao autentificar-se, quando entrar na aplicação pela próxima vez, o *Login* é feito automaticamente.

Se o par *username* / palavra-chave não estiver na base de dados, é apresentada uma mensagem de erro. No caso da autenticação ter sido feita com sucesso, o utilizador é redireccionado para a página principal da sua galeria de conteúdo (Figura 35) ou para a página de onde foi redireccionado quando tentou acedê-la sem estar autenticado.

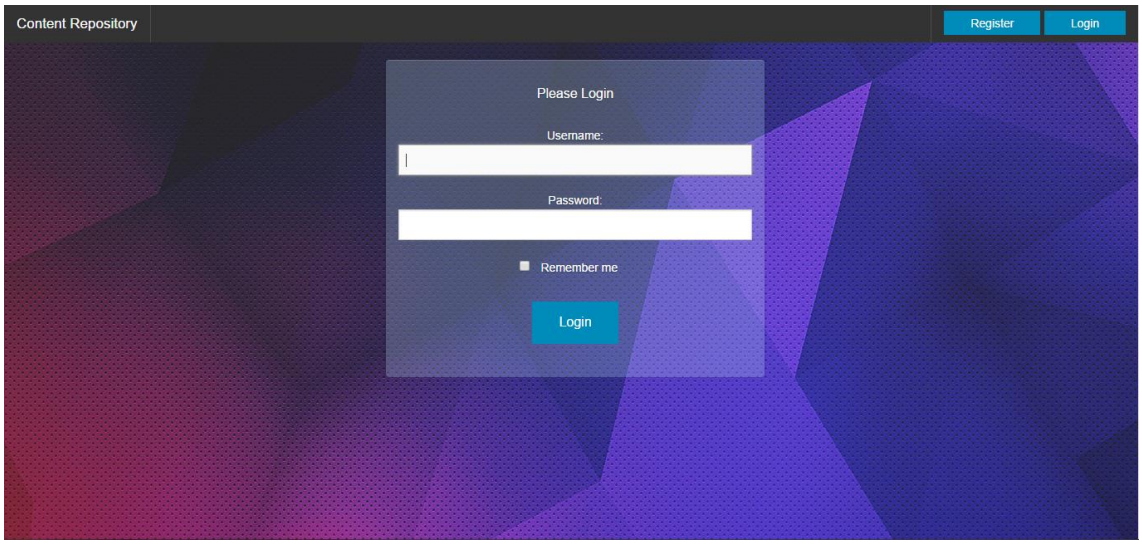

Figura 35 - Página de *Login*

5.3 Página principal de galeria

A figura 36 mostra a página principal de galeria de conteúdos de utilizador. Ao lado esquerdo encontra-se a árvore hierárquica de nós correspondente à estrutura de repositório. Ao expandir a árvore e clicar num dos nós é carregada no lado direito uma grelha com os *thumbnails* de conteúdo correspondente ao nó seleccionado. É de referir que apenas a divisão de galeria é recarregada e o resto de conteúdo da página permanece a mesmo.

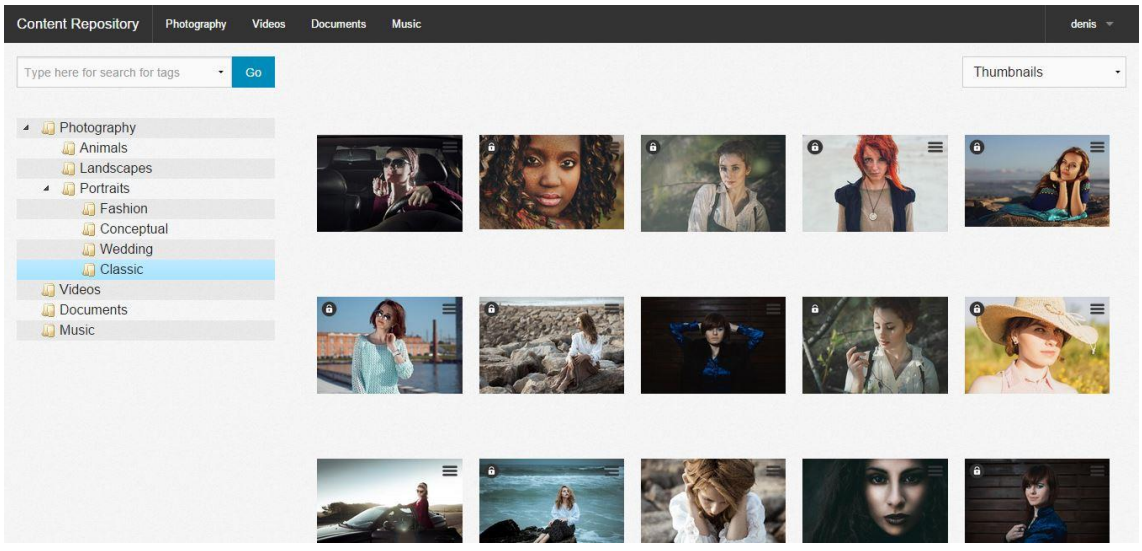

Figura 36 - Página principal de galeria

Cada *thumbnail* na galeria tem um ícone representado por três linhas horizontais no canto superior direito. Quando o utilizador coloca o cursor de rato em cima desse ícone, aparece um menu com as várias operações disponíveis (Figura 37). Para distinguir o conteúdo público de privado, os *thumbnais* de conteúdo privado contêm no canto superior esquerdo um ícone de cadeado.

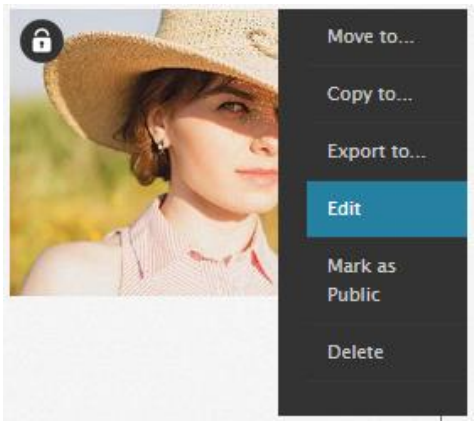

Figura 37 - Menu de acções disponiveis para cada ítem

Ao escolher a opção *Move to* ou *Copy To* é aberta uma janela *pop-up* com a árvore hierárquica, onde o utilizador pode escolher a categoria para onde quer copiar ou mover o ficheiro. (Figura 38).

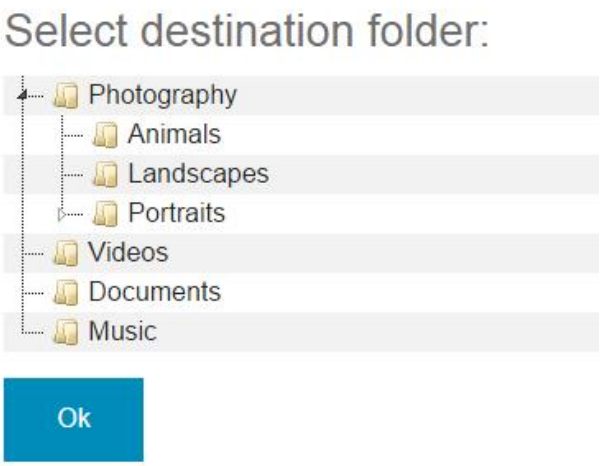

Figura 38 - Janela *pop-up* de escolha de categoria de destino

Para pesquisar pelo conteúdo específico, existe um campo de entrada para pesquisas que funciona com a tecnologia AJAX. A começar introduzir a palavra, são apresentadas as sugestões de *tags* mais utilizados (Figura 39). Além de *tags*, o utilizador pode pesquisar o conteúdo pelo título ou até metadados específicos, introduzindo o tipo de metadados, dois pontos e a seguir o valor pelo qual quer pesquisar. O resultado de pesquisa é apresentado da mesma forma que a galeria.

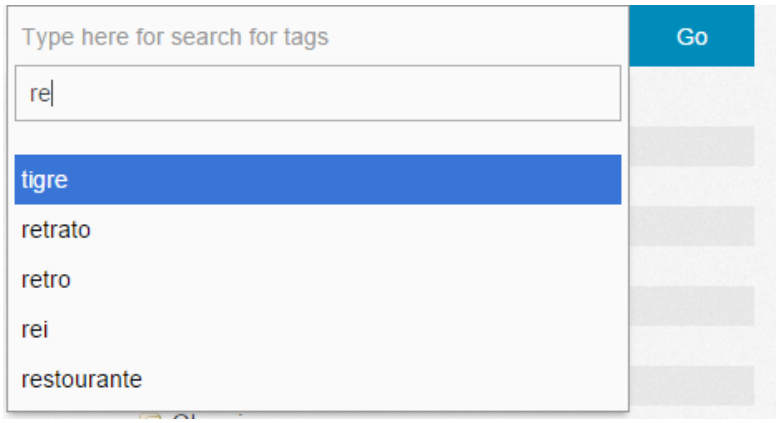

Figura 39 - Campo de pesquisa

Mais um elemento que está presente nesta página é a lista de *dropdown* para escolher o tipo de visualização de galeria, onde o conteúdo pode ser apresentado como uma grelha de *thumbnails* (por defeito) ou como uma tabela.

5.4 Visualização de conteúdo

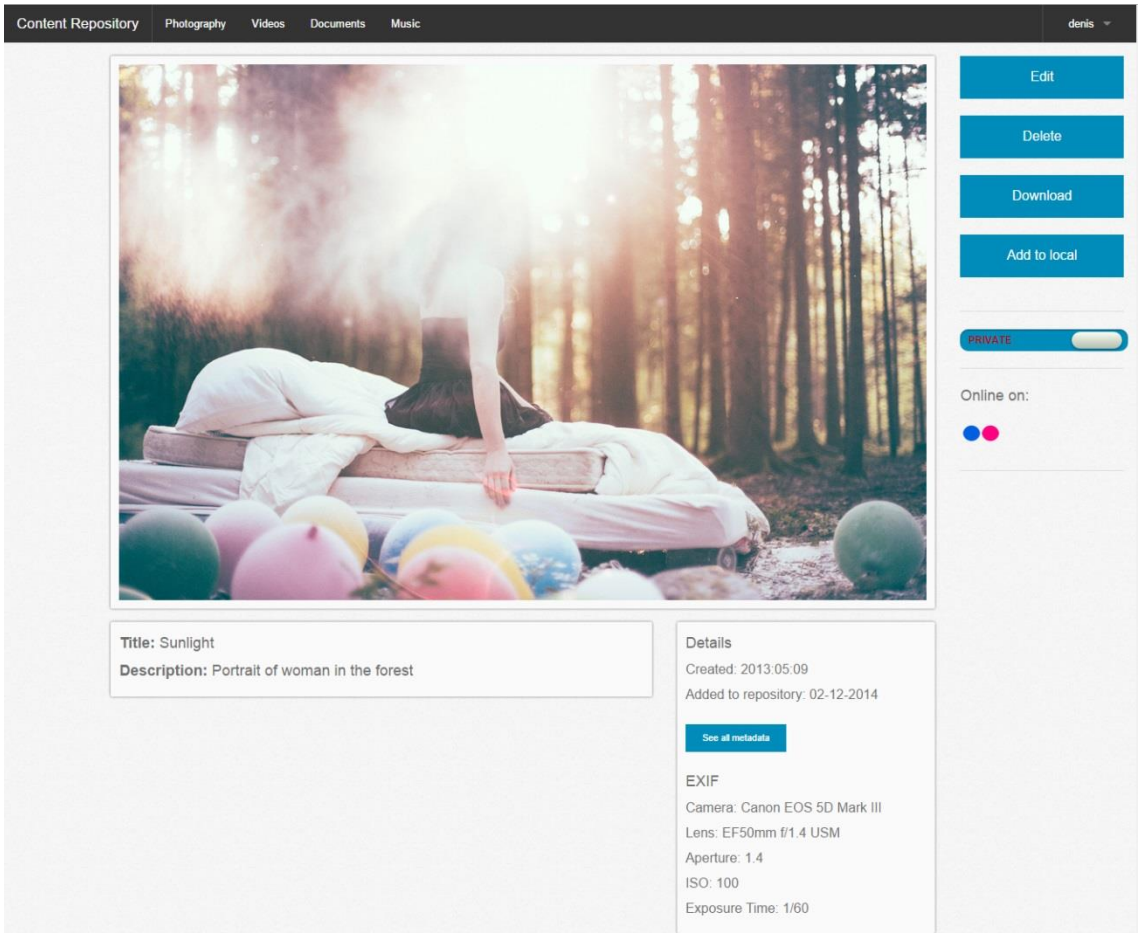

Figura 40 - Visualização de conteúdo seleccionado

Ao seleccionar um item da galeria, é aberta uma janela onde ele é visualizado. (Figura 40). No lado direito encontram-se os botões com as várias operações disponíveis e um *slider* para fazer o conteúdo privado ou público. Além disso, no caso de o conteúdo estiver armazenado *online* no outro serviço, é mostrado o ícone correspondente.

Na parte inferior esquerdo é demonstrado o título e a descrição do conteúdo e na parte inferior direito existe uma lista com os detalhes de conteúdo, como a data de criação, data de colocação no repositório e as definições da máquina digital (se existir).

Mais um elemento da página é o botão que se encontra debaixo da informação sobre o conteúdo. Ao clicar nesse botão é aberta uma janela *pop-up* com uma lista de todos os metadados contidos no conteúdo em questão.

5.5 Árvore hierárquica de nós

Como já foi dito, para organizar o conteúdo é utilizada a árvore hierárquica de nós. O utilizador com o *click* de botão direito do rato pode criar novas categorias, mudar o nome das categorias, apagar as categorias existentes ou copiar e mover as categorias para outro sítio. (Figura 41).

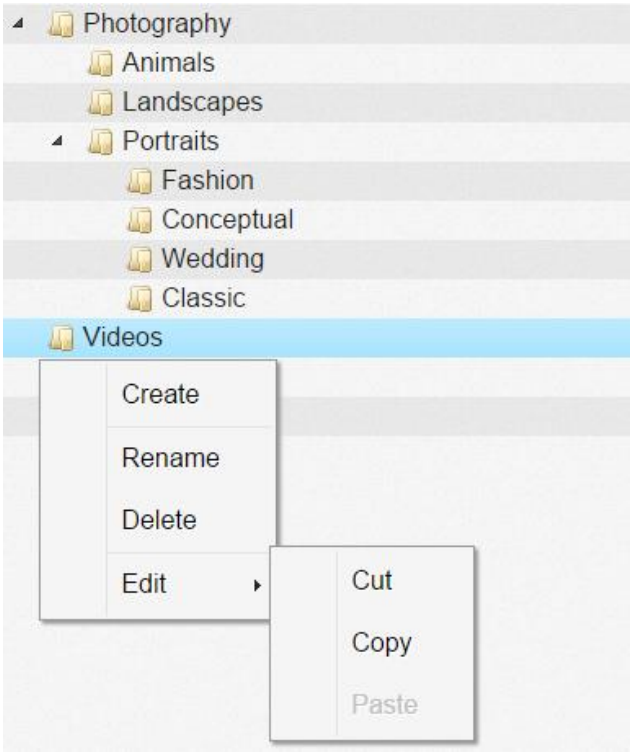

Figura 41 - Menu da árvore hierárquica de nós

5.6 Carregar ficheiros

A página que é responsável pelo *upload* de ficheiros para o repositório contém apenas dois elementos essenciais (Figura 42). No lado esquerdo está situada a árvore hierárquica de nós. É necessário escolher o nó para onde o conteúdo será carregado. No caso contrário, na tentativa de abrir o ficheiro para *upload*, aparece a mensagem de erro a dizer que não foi seleccionado nenhum nó. Ao lado direito da árvore está uma caixa para onde o utilizador pode arrastar o conteúdo de computador ou clicar para escolher os ficheiros a carregar. Suporta o *upload* de múltiplos ficheiros em simultâneo, visualização de tamanho de ficheiro e previsualização de conteúdo de ficheiro. Quando o ficheiro está em processo de *upload*, o processo é visualizado com uma barra que é preenchida em tempo real conforme a percentagem de carregamento de ficheiro (Figura 43). A finalizar o *upload* aparece um ícone de marca de selecção verde no canto superior direito de *thumbnail* de previsualização.

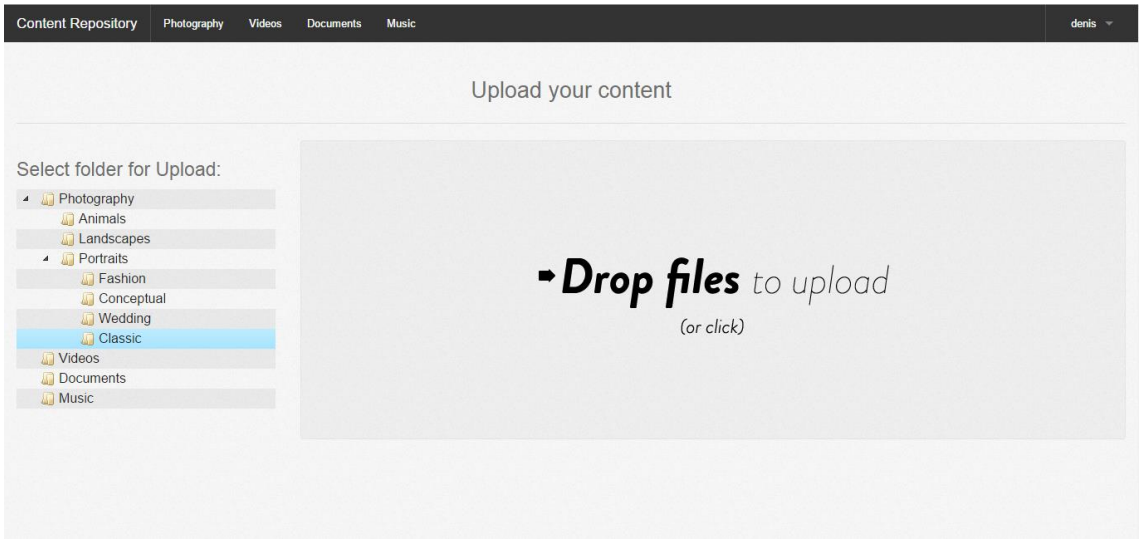

Figura 42 - Página de Upload

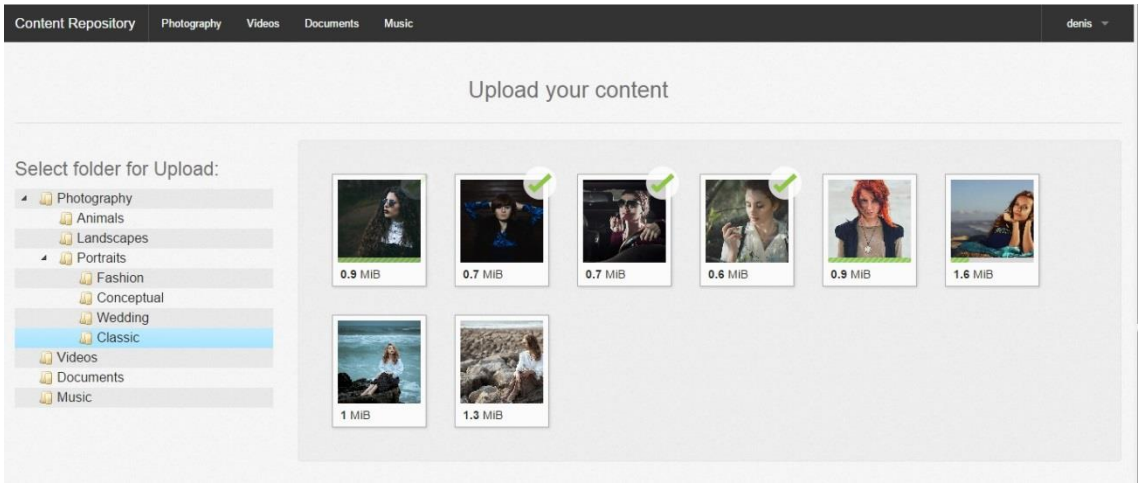

Figura 43 - Página de upload a carregar o conteúdo

5.7 Importar ficheiros

Na página de *import* de ficheiros (Figura 44) também existe a árvore hierárquica de nós para escolher a categoria para onde os ficheiros serão guardados. Depois de escolher um nó, o utilizador tem duas opções. Uma permite importar o conteúdo um por um e outro para importar o conteúdo múltiplo em simultâneo. Para a primeira opção o utilizador precisa de escolher entre as opções possíveis através de uma lista de tipo *dropdown* o serviço de onde deseja importar e na caixa de entrada de baixo colocar o *link* de conteúdo. Para a segunda opção, primeiro o utilizador precisa de escolher o serviço de onde quer importar o conteúdo.

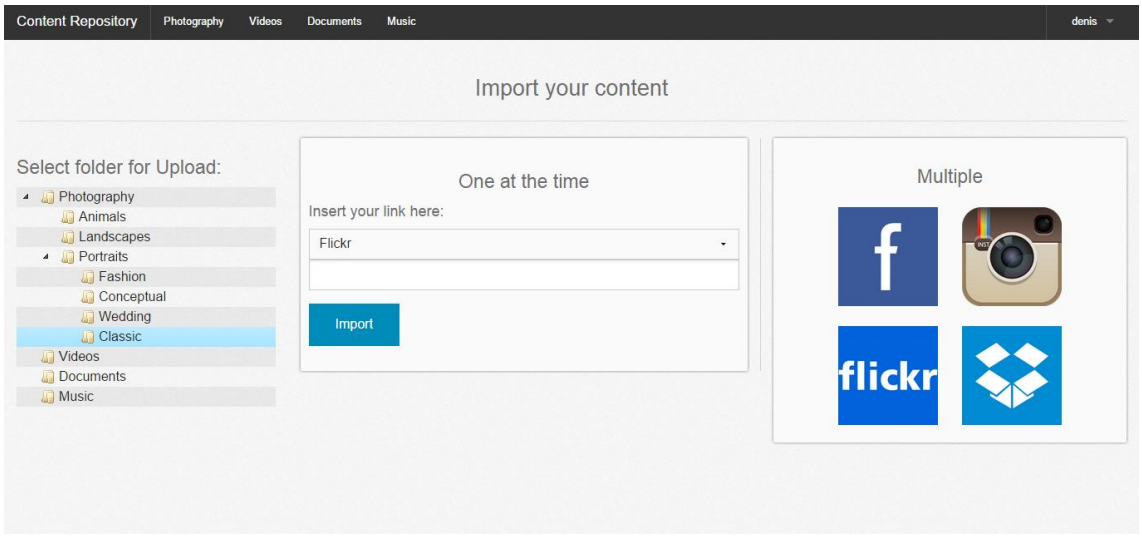

Figura 44 - Página de *Import*

De seguida, é aberta uma janela *pop-up* onde o utilizador pode escrever o nome de utilizador da conta do serviço de origem ou buscar o seu nome se estiver autenticado. (Figura 45). Após a carregar no botão *Get Photos*, é carregada a grelha de *thumbnails* de conteúdo existente na conta do utilizador. Para adicionar o conteúdo para o repositório basta seleccionar os *thumbnails* e clicar no botão *Submit*.

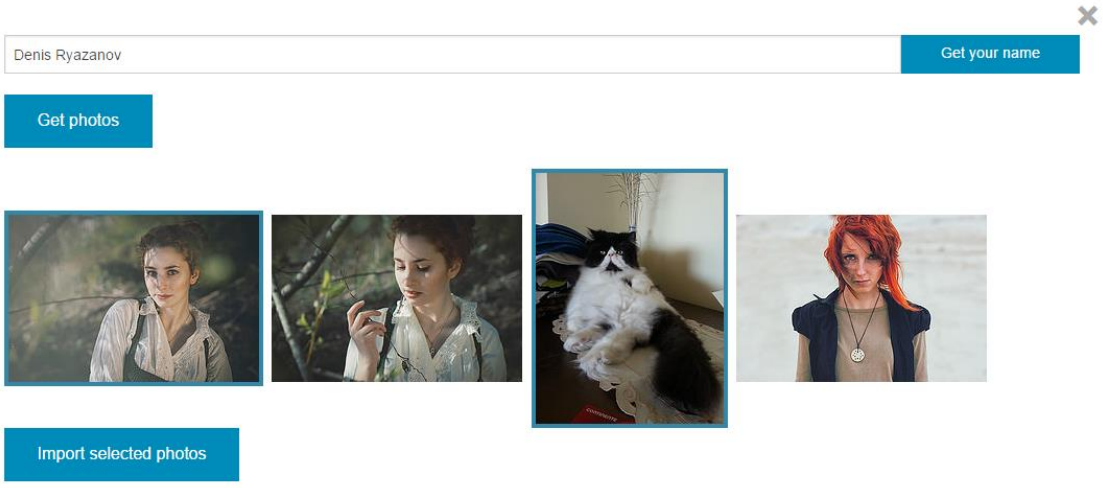

Figura 45 - Página de escolha de conteúdo de Flickr

5.8 Edição de conteúdo

Quando o utilizador escolher a opção de editar algum conteúdo é apresentada uma página que contém a previsualização de conteúdo no lado esquerdo e os campos que o utilizador pode editar no lado direito (Figura 46). Nesses campos é possível editar o título, a descrição, os *tags* e as categorias onde é visualizado esse conteúdo. Para adicionar uma nova categoria, o utilizador precisa de clicar no botão *Add new category* e em baixo é desenrolada a árvore hierárquica de nós onde o utilizador pode escolher a nova categoria. Para remover uma das categoria basta clicar no botão *Remove* a direita da categoria. Depois de editar os campos desejados e clicar no botão *Save*, o utilizador é redireccionado para a página principal de conteúdo.

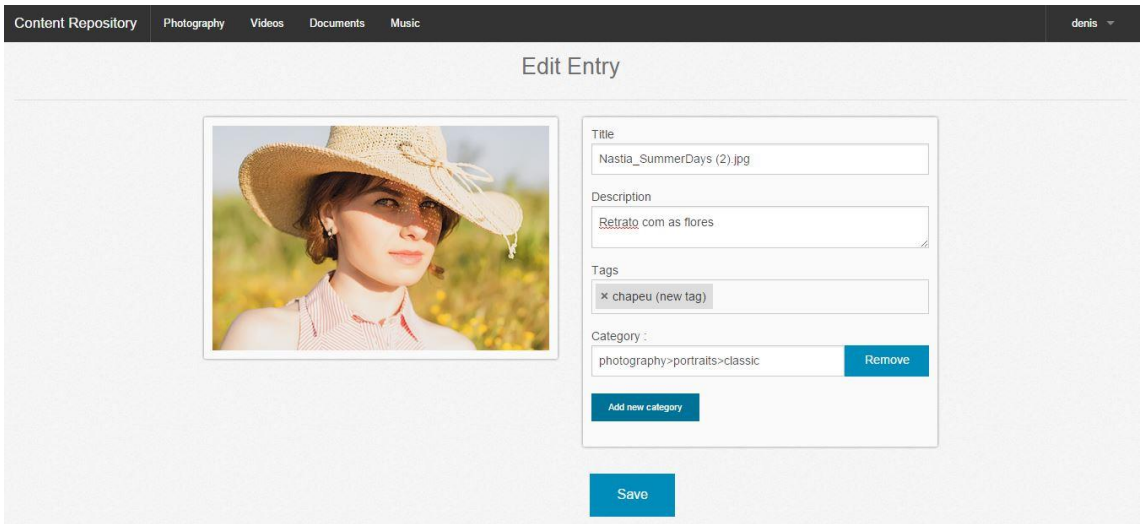

Figura 46 - Página de edição de conteúdo

5.9 *Design* de *Templates*

A página de *Design* de *Templates* (Figura 47) consiste em duas partes. No lado esquerdo existe uma coluna com os *templates* disponíveis. No lado direito, após escolher um dos *templates* é visualizado o esqueleto do *template* escolhido com os blocos pertencentes.

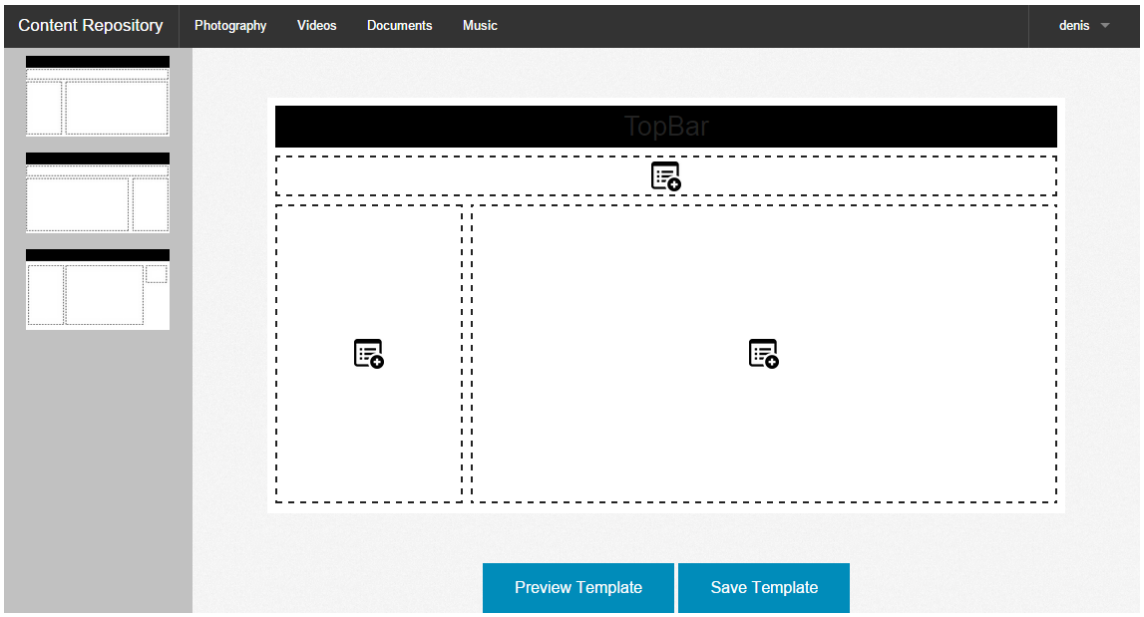

Figura 47 - Página de *design* de *templates*

Ao clicar num dos blocos é aberta uma janela *pop-up* onde o utilizador pode escolher um dos módulos existentes ou criar um novo módulo. (Figura 48).

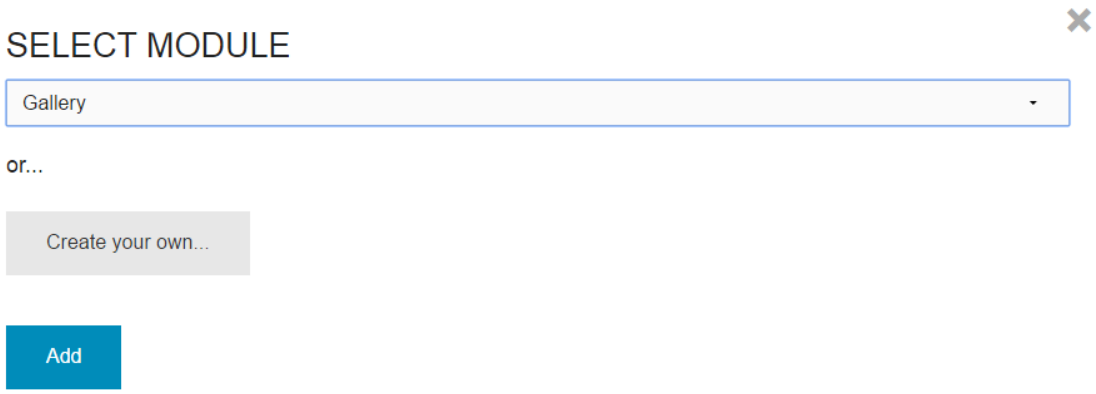

Figura 48 - Janela pop-up de escolha de módulo

Se é clicado o botão *Create your own*, é aberta outra janela *pop-up*, onde é incluído o editor de texto *html* com as várias ferramentas que facilitam o desenho do módulo (Figura 49). É de referir que é possível previsualizar o resultado final durante o desenvolvimento do módulo, pois trata-se de um editor com a capacidade WYSIWYG.

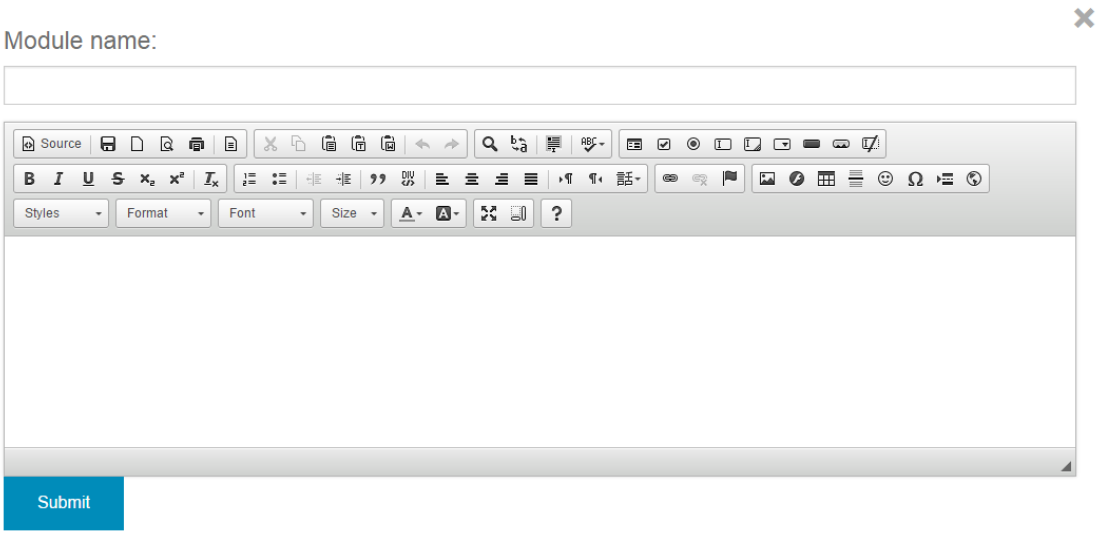

Figura 49 - Janela de editor de texto

Depois de escolher os módulos correspondentes ao cada bloco de *template*, o utilizador pode previsualizar ou guardar o *template* da galeria pública.

5.10 Definições

Todas as definições da aplicação estão disponíveis numa página e estão divididos por separadores. (Figura 50). O primeiro separador contém a informação pessoal do utilizador, o nome, o *email* e o sexo. No segundo separador estão as definições da galeria pública, a escolha de *layout* e um interruptor de activação/desactivação da galeria pública. No terceiro separador o utilizador pode autenticar-se nos serviços externos onde tem a sua conta. E o quarto separador contém a informação do repositório do utilizador, como a data de registo, espaço utilizado, etc.

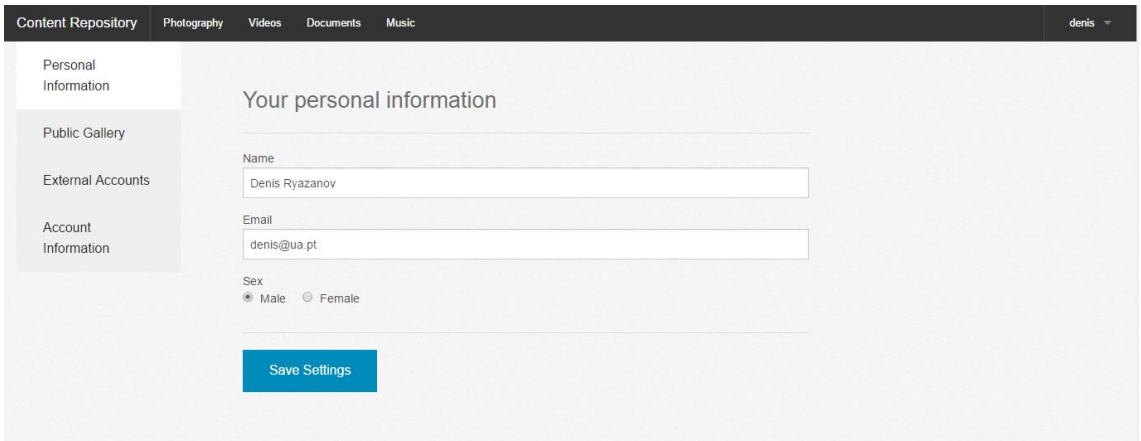

Figura 50 - Página das Definições

5.11 Galeria pública

A galeria pública, dependendo do *template* e *layout* escolhido, pode ter várias aparências. Os únicos blocos obrigatórios são a árvore hierárquica de nós e a galeria de conteúdo. A galeria de conteúdo tem a forma de uma grelha de *thumbnails* dinâmica, onde aparece apenas o conteúdo marcado como público (Figura 51).

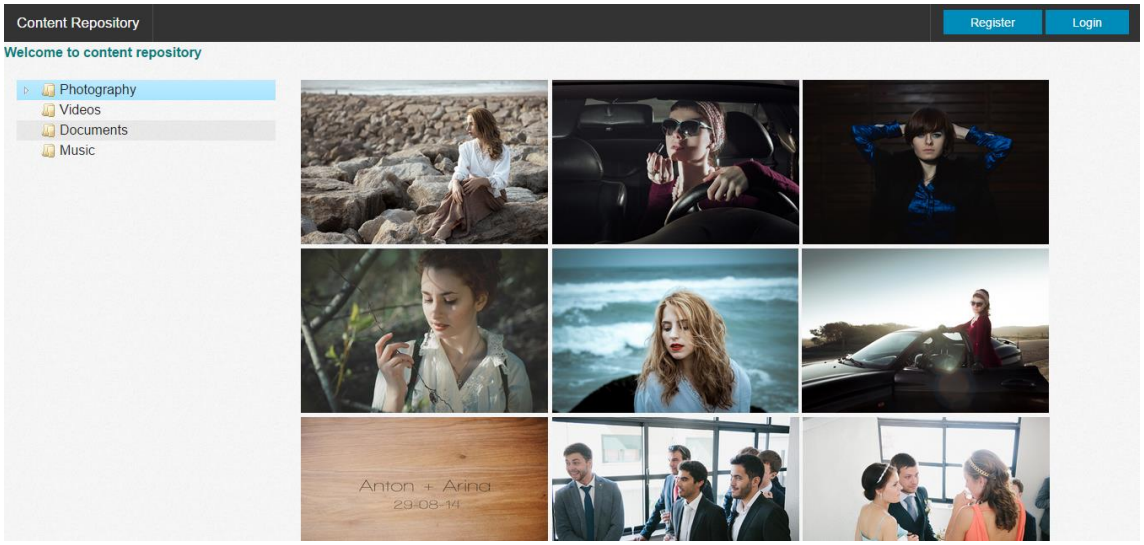

Figura 51- Página de galeria pública do utilizador

Ao clicar num dos *thumbnails*, o ficheiro é estendido conforme o tamanho do ecrã e tem vários elementos de interacção para a navegação, *slideshow* e partilha. (Figura 52).

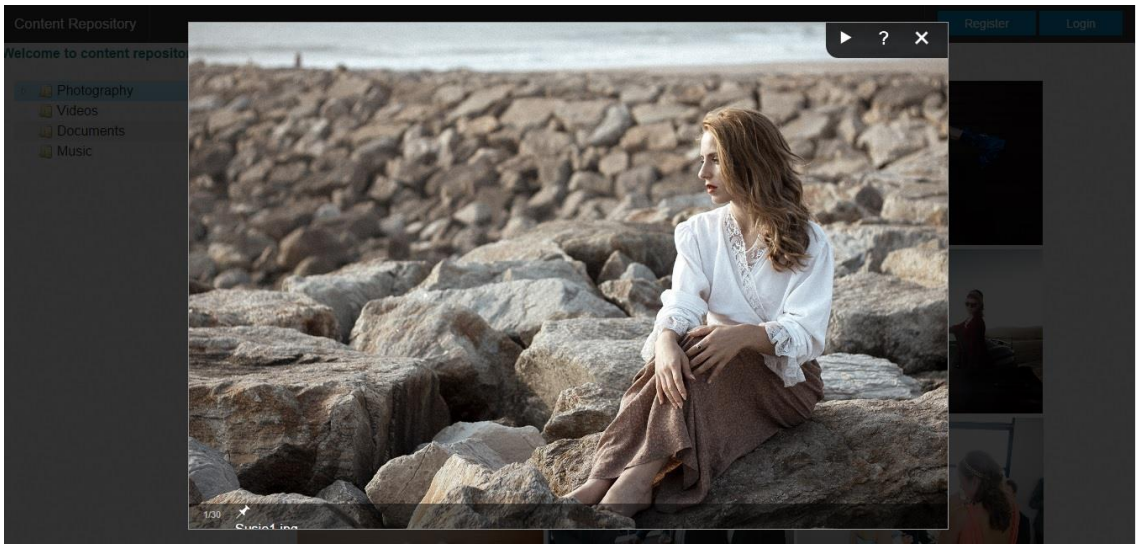

Figura 52 - Visualização de ficheiro escolhido

5.12 Aspecto nos dispositivos móveis

Como já foi referido anteriormente, a aplicação adapta-se dinamicamente aos tamanhos diferentes de ecrãs. Na figura 53 é demonstrado o aspecto das várias páginas da aplicação utilizando um *smartphone.*

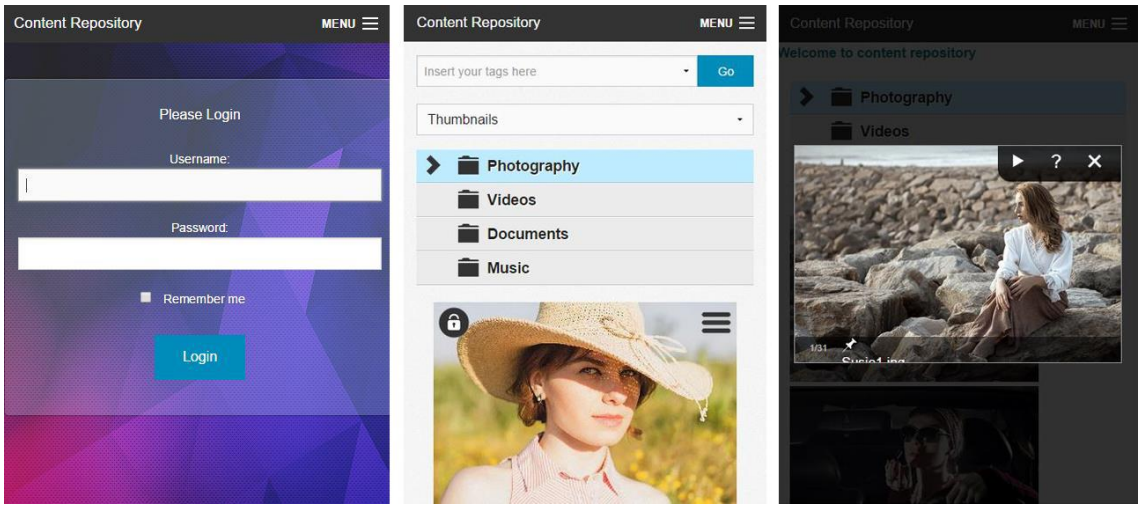

Figura 53 - Aspecto da aplicação no *smartphone*

6. Conclusão

Como foi verificado, o número de aplicações que têm como função o armazenamento e a partilha de conteúdos, tem aumentado, dificultando assim a gestão e organização de conteúdos que ficam espalhados na Internet, no computador e outros dispositivos do utilizador.

No entanto a maioria de serviços e aplicações existentes tem certas limitações. Alguns permitem armazenar os ficheiros de qualquer tipo mas sem a possibilidade de partilhar esse conteúdo publicamente como galeria. Outros têm a funcionalidade de galeria, mas permitem armazenar apenas certo tipo de ficheiros. A outra desvantagem de muitos serviços é espaço limitado onde para ter o espaço adicional é necessário adquiri-lo com um valor monetário, que às vezes chega a valores elevados.

Para resolver essas limitações foi desenvolvida em Grails e Groovy a aplicação genérica que foi o objectivo desta dissertação. Este documento apresenta a descrição teórica do projecto, estudo das tecnologias usadas durante o desenvolvimento, arquitectura da aplicação e demonstração da mesma. Todos os objectivos definidos na modelação da aplicação foram alcançados.

A aplicação pode ser usada de maneiras diferentes respondendo as necessidades de cada utilizador, tem uma interface simples e intuitiva, permite armazenar os formatos de ficheiros multimédia mais comuns, tem integração com os outros serviços existentes, permite armazenar e gerir os ficheiros, tanto no repositório como os que se encontram nos serviços externos, e suporta as pesquisas de conteúdo por *tags*, título e metadados. O utilizador tem a possibilidade de partilhar o seu conteúdo na forma de uma galeria pública que pode ser personalizada consoante o gosto pessoal.

Mais uma vantagem da aplicação desenvolvida é o uso do repositório de conteúdo Apache Jackrabbit, o que permite armazenar todo o conteúdo e a informação correspondente de uma forma independente de sistemas operativos e de base de dados.

6.1 Trabalhos futuros

Apesar de todos os objectivos alcançados, existem ainda várias funcionalidades que podem ser adicionadas no futuro para melhorar a aplicação.

Uma delas é utilização de *namespaces* hierárquicos nos *tags*, uma funcionalidade que foi implementada durante esse trabalho mas acabou de ser desactivada na aplicação final pela razão de complexidade de visualização de estrutura de *tags* na aplicação. A funcionalidade trata-se de ter uma estrutura hierárquica para *tags* que permite diferenciar os *tags* iguais que significam assuntos diferentes. Um exemplo deste caso é uma vela, que pode ser tanto a vela de um barco como a vela de parafina – uma fonte da luz. Neste caso para a vela de barco temos por exemplo uma hierarquia seguinte “transporte->barco->vela”, onde a vela pertence

ao barco e o barco pertence ao transporte. Entretanto todos os três são os objectos da classe Tag.

Outra funcionalidade que pode melhorar a aplicação é fazer uma galeria que contém os conteúdos públicos de todos os utilizadores.

Também uma funcionalidade útil para globalização da aplicação pode ser a internacionalização. Utilizando as ferramentas de Grails é possível facilmente fazer a aplicação em idiomas diferentes.

Por fim, como já foi dito na introdução, seria útil disponibilizar o projecto desenvolvido para o público para a integração nos projectos futuros. Para isso basta guardar o projecto como *plugin* para Grails.

6.2 Aprendizagem

Ao longo deste trabalho foi necessário obter os conhecimentos profundos acerca de processo de desenvolvimento de aplicações Web. Foi utilizado um número grande de tecnologias que foi um ponto positivo para obter muitos conhecimentos e ganhar as novas experiências na construção de aplicações. A nível de linguagens de programação foram usados:

-HTML

-CSS

-Groovy

-Ajax

-JavaScript

-jQuery

-Java

-The Hibernate Query Language (HQL)

Também foram usadas as seguintes ferramentas:

-Grails

-Spring Security

7. Referências

[1] "Groovy", Disponível em http://groovy.codehaus.org/. Acedido em 3 de Setembro de 2014

[2] "Grails", Disponível em https://grails.org/. Acedido em 3 de Setembro de 2014

[3] "Hibernate", Disponível em http://hibernate.org/. Acedido em 5 de Setembro de 2014

[4] "Spring", Disponível em http://spring.io/. Acedido em 5 de Setembro de 2014.

[5] "SiteMesh", Disponível em http://wiki.sitemesh.org/wiki/display/sitemesh/Home. Acedido em 5 de Setembro de 2014.

[6] "Tomcat", Disponível em http://tomcat.apache.org/. Acedido em 5 de Setembro de 2014.

[7] "H2", Disponível em http://www.h2database.com/html/main.html. Acedido em 5 de Setembro de 2014.

[8] "JSR 283: Content Repository for JavaTM Technology API Version 2.0", Disponível em https://jcp.org/en/jsr/detail?id=283. Acedido em 9 de Outubro de 2014.

[9] "JSR 333: Content Repository API for Java Technology 2.1", Disponível em https://jcp.org/en/jsr/detail?id=333. Acedido em 9 de Outubro de 2014.

[10] "Apache Jackrabbit", Disponível em http://jackrabbit.apache.org/. Acedido em 2 de Setembro de 2014.

[11] "Apache Lucene", Disponível em http://lucene.apache.org/core/ Acedido em 2 de Setembro de 2014.

[12] Camera & Imaging Products Association. "Exchangeable image file format for digital still cameras:Exif Version 2.3". 2010. Disponível em http://www.cipa.jp/std/documents/e/DC-008-2012_E.pdf. Acedido em 11 de Outubro de 2014.

[13] "Machine Readable Cataloging", Disponível em http://www.loc.gov/marc/. Acedido em 25 de Setembro de 2014.

[14] "The Dublin Core Metadata Initiative", Disponível em http://dublincore.org/. Acedido em 25 de Setembro de 2014.

[15] "The MP3 Tag Standart", Disponível em http://id3.org/. Acedido em 25 de Setembro de 2014.

[16] "Apache Tika - a content analysis toolkit", Disponível em http://tika.apache.org/. Acedido em 27 de Setembro de 2014.

[17] "Dropbox", Disponível em https://www.dropbox.com/. Acedido em 28 de Agosto de 2014.

[18] “Picturelife”, Disponível em https://picturelife.com. Acedido em 29 de Agosto de 2014.

[19] “Flickr”, Disponível em https://www.flickr.com/. Acedido em 28 de Agosto de 2014.

[20] “Google Drive”, Disponível em https://drive.google.com. Acedido em 28 de Agosto de 2014.

[21] “Trovebox”, Disponível em https://trovebox.com/. Acedido em 29 de Agosto de 2014.

[22] “Spring Security”, Disponível em http://projects.spring.io/spring-security/. Acedido em 1 de Outubro de 2014.

[23] “Spring Security Core Plugin”, Disponível em http://grails.org/plugin/spring-security-core. Acedido em 1 de Outubro de 2014.

[24] “Zurb Foundation”, Disponível em http://foundation.zurb.com/. Acedido em 4 de Outubro de 2014.

[25] “jsTree”, Disponível em http://www.jstree.com/. Acedido em 4 de Outubro de 2014.

[26] “Select2”, Disponível em http://ivaynberg.github.io/select2/. Acedido em 4 de Outubro de 2014.

[27] “Dropzone.js”, Disponível em http://www.dropzonejs.com/. Acedido em 4 de Outubro de 2014.

[28] “Yoxigen YoxView”, Disponível em http://www.yoxigen.com/yoxview/. Acedido em 4 de Outubro de 2014.

[29] “CKEditor”, Disponível em http://ckeditor.com/. Acedido em 4 de Outubro de 2014.

[30] “Mega”, Disponível em https://mega.co.nz/ Acedido em 17 de Outubro.

[31] Jim Shingler, Joseph Faisal Nusairat , Christopher M Judd. Beginning Groovy and Grails From Novice to Professional. 2008.

[32] Scott Davis, Jason Rudolph. Getting Started with Grails, Second Edition. 2010.

[33] Bashar Abdul-Jawad. Groovy and Grails Recipes (Expert's Voice in Open Source). 2008.

[34] “OAuth”, Disponível em http://oauth.net/. Acedido em 22 de Outubro de 2014.

[35] “Using OAuth with Flickr”, Disponível em https://www.flickr.com/services/api/auth.oauth.html. Acedido em 24 de Outubro de 2014.

[36] Hugo Feitosa de Figueirêdo, Yuri Almeida Lacerda, Anselmo Cardoso de Paiva, Marco Antonio Casanova, Cláudio de Souza Baptista. PhotoGeo: a photo digital library with spatial-temporal support and self-annotation. 2011. Disponível em http://link.springer.com/article/10.1007%2Fs11042-011-0745-x. Acedido em 30 de Outubro de 2014.

[37] Jim Gemmell, Gordon Bell, Roger Lueder. MyLifeBits: a personal database for everything. Disponível em http://dl.acm.org/citation.cfm?doid=1107458.1107460. Acedido em 30 de Outubro de 2014.

[38] "JSR 170: Content Repository for JavaTM technology API", Disponível em https://jcp.org/en/jsr/detail?id=170. Acedido em 9 de Outubro de 2014.

[39] "500px", Disponível em https://500px.com. Acedido em 5 de Novembro de 2014.

[40] "Youtube", Disponível em https://www.youtube.com/. Acedido em 5 de Novembro de 2014.

[41] "Vimeo", Disponível em https://vimeo.com/. Acedido em 5 de Novembro de 2014.

[42] "Joomla", Disponível em http://www.joomla.org/. Acedido em 17 de Novembro de 2014

[42] "Drupal", Disponível em https://www.drupal.org/. Acedido em 17 de Novembro de 2014

[42] "Wordpress", Disponível em https://wordpress.org/. Acedido em 17 de Novembro de 2014

[42] "Blogspot", Disponível em https://www.blogger.com/. Acedido em 17 de Novembro de 2014

[42] "Magnolia", Disponível em http://www.magnolia-cms.com/. Acedido em 17 de Novembro de 2014

[42] "Hippo CMS", Disponível em http://www.onehippo.com/. Acedido em 17 de Novembro de 2014

[43] Jim Gemmell, Gordon Bell. "A digital Life", SCIENTIFIC AMERICAN, p.58-65, Março de 2007.

[44] "Google Calendar", Disponível em https://www.google.com/calendar. Acedido em 22 de Novembro de 2014.

[45] "Wikimapia", Disponível em http://wikimapia.org/. Acedido em 22 de Novembro de 2014.

[46] "GeoNames", Disponível em http://www.geonames.org/. Acedido em 22 de Novembro de 2014.